DURHAM UNIVERSITY

MASTERS THESIS

# Automatic Update of Airport GIS by Remote Sensing Image Analysis

*Author:*
Philip JACKSON

*Supervisor:*
Dr. Boguslaw OBARA

*A thesis submitted in fulfilment of the requirements for the degree of Masters by Research*

*in the*

Innovative Computing Group
Engineering and Computing Sciences

June 2, 2016

# Declaration of Authorship

I, Philip JACKSON, declare that this thesis titled, "Automatic Update of Airport GIS by Remote Sensing Image Analysis" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

DURHAM UNIVERSITY

# *Abstract*

Faculty of Science
Engineering and Computing Sciences

Masters by Research

**Automatic Update of Airport GIS by Remote Sensing Image Analysis**

by Philip JACKSON

This project investigates ways to automatically update Geographic Information Systems (GIS) for airports by analysis of Very High Resolution (VHR) remote sensing images. These GIS databases map the physical layout of an airport by representing a broad range of features (such as runways, taxiways and roads) as georeferenced vector objects. Updating such systems therefore involves both automatic detection of relevant objects from remotely sensed images, and comparison of these objects between bi-temporal images. The size of the VHR images and the diversity of the object types to be captured in the GIS databases makes this a very large and complex problem. Therefore we must split it into smaller parts which can be framed as instances of image processing problems. The aim of this project is to apply a range of methodologies to these problems and compare their results, providing quantitative data where possible. In this report, we devote a chapter to each sub-problem that was focussed on.

Chapter 1 begins by introducing the background and motivation of the project, and describes the problem in more detail.

Chapter 2 presents a method for detecting and segmenting runways, by detecting their distinctive markings and feeding them into a modified Hough transform. The algorithm was tested on a dataset of six bi-temporal remote sensing image pairs and validated against manually generated ground-truth GIS data, provided by Jeppesen.

Chapter 3 investigates co-registration of bi-temporal images, as a necessary precursor to most direct change detection algorithms. Chapter 4 then tests a range of bi-temporal change detection algorithms (some standard, some novel) on co-registered images of airports, with the aim of producing a change heat-map which may assist a human operator in rapidly focussing attention on areas that have changed significantly.

Chapter 5 explores a number of approaches to detecting curvilinear AMDB features such as taxilines and stopbars, by means of enhancing such features and suppressing others, prior to thresholding. Finally in Chapter 6 we develop a method for distinguishing between AMDB lines and other curvilinear structures that may occur in an image, by analysing the connectivity between such features and the runways.

# *Acknowledgements*

This project was sponsored by a research grant from Jeppesen GmbH, grant number RF081BO. I would like to thank my supervisor for his advice and support with this thesis.

# Contents

# List of Figures

# List of Tables

# List of Abbreviations

| | |
|---|---|
| **AMDB** | **A**irport **M**apping **D**ata**B**ase |
| **GIS** | **G**eographic **I**nformation **S**ystem |
| **HT** | **Hough Transform** |
| **CD** | **C**hange **D**etection |
| **SURF** | **S**peeded-**U**p **R**obust **F**eatures |
| **VHR** | **V**ery **H**igh **R**esolution |
| **SIFT** | **Scale Invariant Feature Transform** |
| **RGB** | **R**ed **G**reen **B**lue |
| **FOD** | **F**oreign **O**bject **D**etection |
| **ROI** | **R**egion **O**f **I**nterest |
| **HSV** | **H**ue **S**aturation **V**alue |
| **RANSAC** | **RAN**dom **SA**mple **C**onsensus |
| **QGIS** | **Q**uantum **GIS** |
| **FREAK** | **F**ast **RE**tin**A** **K**eypoint |
| **FAST** | **F**eatures from **A**ccelerated segment **T**est |
| **BRISK** | **B**inary **R**obust **I**nvariant **S**calable **K**eypoints |

# Chapter 1

# Introduction

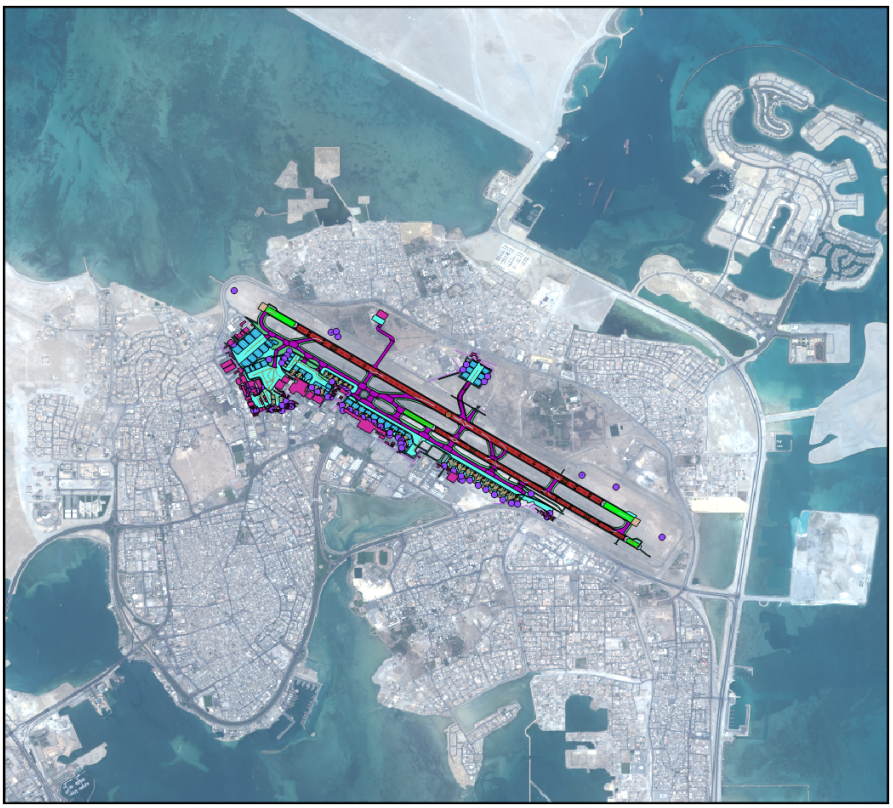
As the volume of international trade and travel has increased over the years, airports have likewise grown into very large and complex structures, where thousands of individuals perform complex and interacting activities. Many of these individuals, from pilots and air traffic controllers to surface vehicle operators and maintenance crews, require knowledge of the airport's spatial layout. The industrial sponsor of this project, Jeppesen GmbH (a Boeing company), provides this information in the form of an Airport Mapping DataBase (AMDB), a type of GIS data. An AMDB is a collection of polygons, lines and points, each specified in terms of physical longitude/latitude coordinates and each labelling some significant surface feature on the airport. For example, lines are used to label the centrelines of runways and taxiways, while polygons mark out aprons and standing areas. The AMDB is stored as a collection of Esri™ shapefiles, a widely supported format allowing interoperability between ArcGIS (the system in which they are created) and other GIS software products.

Currently these shapefiles are produced manually by a team of human experts, who analyse a Very High Resolution (VHR) satellite image of the airport and then annotate the image with vector objects. The image itself has a resolution of 61cm per pixel and is acquired by the QuickBirdII satellite, the same platform which supplies some of Google Earth's data. This image is also georeferenced - a separate pre-processing stage in which the pixels are mapped to physical longitude/latitude coordinates. This facilitates the AMDB team to create the shapefiles (which have physical coordinates) by clicking on the image, and allows the shapefiles to later be graphically overlaid on the image in any GIS viewer.

Due to the volume and complexity of information in a single AMDB, AMDB creation is a slow and painstaking process. Even more time consuming is AMDB maintenance, since this is an ongoing process which must be performed periodically for every AMDB, of which there are hundreds. Updating an AMDB involves similar processes to creating one, since in both cases the relevant airport features must be recognised and described from an airport image. However, updating an AMDB is also a problem of *change detection*, since if no change has occurred to the airport then no change need be made to the AMDB. Therefore, some element of comparison between the old and new image of the airport is necessary. This thesis investigates the feasibility of several methods to automate both feature detection and comparison, in order to alleviate the time cost of AMDB creation and maintenance.

The size and complexity of the satellite images involved (typically of the order 10000x10000 pixels), and of the features themselves, makes this a very challenging problem from an image processing standpoint. Many other image processing applications typically have a single, consistent pattern to extract from a carefully prepared specimen. For example, one may wish to extract blood vessels from a retinal image, recognise characters in a scanned document, or classify a tissue sample as cancerous or

(A) Bahrain International Airport



(B) Bahrain International Airport with GIS data overlaid

FIGURE 1.1: A QuickBirdII image of Bahrain International Airport, with GIS data overlaid in (b). Rendered in QGIS, an open-source GIS package.

non-cancerous. These are already difficult and actively researched problems, but full generation of an AMDB from a satellite image adds additional challenges, mostly due to the scale of the problem. Specifically:

- AMDBs contain a wide variety of features, all of which we would like to detect but many of which require different approaches.
- There is considerable variation in the appearance of these features within any given class, even within the same airport.
- The images we need to process are very large and contain huge amounts of irrelevant information. Therefore, preventing false positives is just as important as detecting the features that we are searching for.

The broad and multi-faceted nature of the problem has influenced the course of our project. Rather than focussing solely on a single technique, we address several subproblems, exploring multiple ways in which automation can make it easier to create and update AMDBs. For example, while automatic detection of objects and generation of shapefiles is the most obvious approach to take, it is also possible to develop software which would merely assist a human operator, allowing them to complete the same task in less time. Such approaches may potentially be very fruitful, and are explored in Chapters 3 and 4. The varied approach we have taken is reflected in the structure of this thesis, each chapter covering a different technique or family of techniques. Chapter 2 details a method for detecting and segmenting runways, which was presented in the ISPA 2015 conference. Chapter 3 investigates image co-registration as a precursor step to direct change detection. Chapter 4 then applies a number of existing change detection methods to our dataset, and tests some novel approaches - in particular, we develop a new contrast invariant change detection technique called Local Ternary Patterns. Chapter 5 examines methods for enhancing curvilinear features in an image, with a view to detecting line type AMDB features, such as taxilines and stopbars. Finally, Chapter 6 demonstrates a proof-of-concept for a method to detect only relevant AMDB lines, ignoring other curvilinear features that might be present in the image.

# Chapter 2

# Runway Detection

## 2.1 Introduction

Runways are unique and defining features of airports. This is significant, because in many of the images in our dataset, the airport itself occupies less than half the area of the image. It is therefore advantageous to know where the runways are, since arguably they form the core of an airport, in the sense that all taxiways and aprons must eventually connect to a runway. Knowing their position, one can use the runways as an anchor point from which to search for other airport features. This effectively narrows the search space when detecting objects of interest, and could make it easier to discard false positives if they occur too far from the runways. Therefore, although the runways themselves are only built or extended very rarely, an automatic detection capability is still desirable, if only as the precursor to detecting other features. For example, one may wish to detect runway markings, since they are part of the AMDB feature set and may be updated more regularly. By defining the search space to be the runway's area, the problem is reduced to finding solid bright shapes on a smooth dark background. This research has been published in the proceedings of the 9th International Symposium on Image and Signal Processing and Analysis [1].

## 2.2 Background

### 2.2.1 Related Work

Due to its importance in aviation and military operations, there have been several prior studies on automatic runway detection. Han *et al.*[2] segmented runways from low resolution images by using edge geometry to find long, homogeneous rectangles that were brighter than their surroundings. Tandra *et al.*[3] produced an edge detection and thresholding algorithm for use as part of a Foreign Object Debris (FOD) detection system. A real-time FOD detection system was developed by Kniaz *et al.*[4], which processes input from an infra-red camera mounted on the front of a plane; this system detected the runway's side-stripes and threshold markings by binarizing the gradient image with Otsu's method, and choosing connected components subject to a set of assumptions about the runway's structure.

Yang *et al.*[5] proposed a method that employed the Hough Transform (HT) in conjunction with Otsu thresholding and fractional gradient edge detection to find runways, but was unable to distinguish between runways and other straight objects.

These methods take diverse approaches but all share a few common assumptions: that the runways are long, straight objects of uniform intensity and that these criteria distinguish a runway from other features in the image.

Alternatively, Aytekin *et al.*[6] proposed a texture-based runway detection algorithm that used the Adaboost machine learning package to identify 32x32 pixel image tiles as runway or non-runway. The algorithm could learn from its training data which of 137 possible texture features were the most salient indicators, and construct its own classification scheme based on those.

Huertas *et al.*[7] detected runways in aerial photography using a hypothesize and test framework, demonstrating how difficult the task can become when the runways are not simply dark or bright rectangles. Hypotheses were formed by matching together pairs of long parallel edges, and verified by checking for runway markings between them. A complex sequence of decisions is followed to detect segments, join them when they are broken, disregard redundant information and to accept or reject hypothetical runways and markings, based on much prior knowledge of the physical structure of the runways. The process is demonstrated to work on two extremely complex airport images, but would be difficult to replicate due to its complexity and many parameters which may have been overfit to the two images it is tested on.

### 2.2.2 Limitations of Traditional Methods

VHR images of entire airports (and their surroundings) present challenges not fully addressed in the above papers. Unlike in the case of FOD systems, which monitor a single runway, an unknown number of runways are present in remote sensing images, along with many irrelevant features such as roads and residential areas. In lower resolution data, runways can appear as bright rectangles, surrounded by largely flat terrain; however, in the case of VHR remote sensing imagery, the background and the runways themselves become much more complex. We also find that runways may be brighter or darker than their surroundings, or even change intensity along their length if different materials have been used for construction or parts of the runway have been renewed or extended. This makes intensity thresholding, as relied on in several previous studies (e.g. [2][5][8]), difficult to apply to these images.

The most obvious distinguishing feature of runways is that they are generally the longest, straightest objects present in an airport. The HT is the most obvious approach to extracting such an object, and has been applied to runway detection in the past [8]. The standard HT method [9] identifies the most prominent lines in an image by means of exhaustive search through a "Hough space", consisting of all possible straight lines subject to some quantisation in their parameters. These parameters, typically, are the line's closest approach to the origin, $\rho$, and the angle between the line's normal vector and the $x$-axis, $\theta$. The Hough space itself is a matrix (sometimes called an "accumulator matrix"), in which each cell corresponds to a line with different parameters. The value in each cell is the number of edge pixels intersected by that corresponding line. Once these values have been computed, prominent lines may be extracted by detecting peaks in the Hough space. Since the standard HT detects straight lines rather than rectangles of non-zero width, the simplest way to apply it to runway detection is to produce a binary edge map, for example from the Canny [10] algorithm, and run the HT on that. In theory the HT will extract from it the long edges of the runways, as they should form the most prominent lines in the image.

This naive approach is severely limited, ultimately because the HT works by choosing the lines which pass through the most edge pixels. The dominant detected signals are often not runway edges, for several reasons:

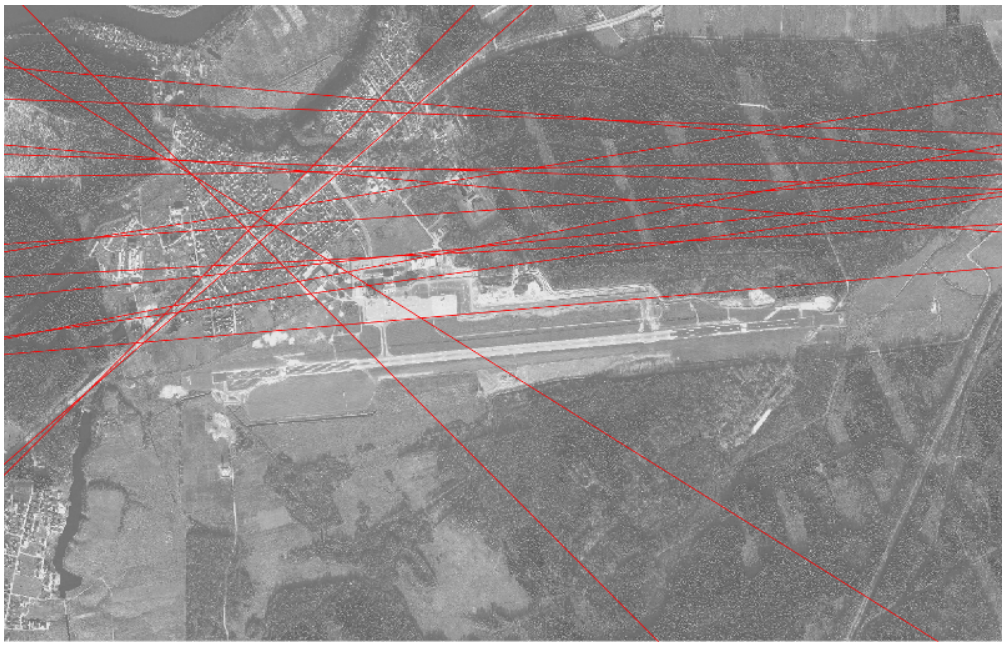


FIGURE 2.1: Standard Hough Transform calculated from a Canny edge map of Kaunas International Airport. Lines (shown in red) are drawn preferentially through trees, since they produce many edge pixels in high resolution data.

- Many other objects in the image, such as taxiways, roads, tyre marks down the centre of many runways and runway markings themselves, present competing edge pixels.
- The edges of runways are usually broken by adjoining taxiways, reducing the number of edge pixels they produce in the edge map.
- Regions which in lower resolution data may appear flat, such as trees and residential areas, produce many edge pixels in VHR data. Lines crossing these regions may intersect more edge pixels than lines along any runway edge, causing the HT to preferentially return these meaningless lines (see Figure 2.1).
- When long edges (thousands of edge pixels long) are being searched for, a very high angular resolution must be used. This is because in order for the edge to cause a tall peak in the Hough space, one of the lines the HT checks must match that edge perfectly, passing through at least most of its edge pixels. With long lines, this can only happen when a checked line has a very similar angle to the runway edge, or else the lines will diverge.

Even when we down-sampled the images to reduce the edge pixel contribution from high spatial frequency regions, we still found it surprisingly difficult for the HT to detect runway edges reliably. A core limitation of the standard HT is that it is unable to automatically detect the number of lines; one can only extract the $n$ most prominent peaks in the Hough space. This is particularly problematic in the case of previously unseen (or altered since last inspection) airports that may contain an unknown number of runways. The only non-arbitrary way to specify $n$ is to specify a minimum number of edge pixels a line must have to qualify as a line, and extract all Hough space peaks with a height above that value. Since the number of edge pixels along runway edges is so variable though, this threshold is difficult to choose satisfactorily. Even when such a threshold is set, the runway edges will only be the only lines returned if there are

no other lines more prominent in the Hough space, which is not generally the case. Therefore, the HT can never be guaranteed to return only runway edges, and the lines it does return can only be treated as hypothetical runways. Usually, with sufficient parameter tweaking and image pre-processing, it is possible to extract at least one edge per runway, but even this is not guaranteed.

## 2.3 Method

Our solution is a way of circumventing the problems caused by image complexity, so that the HT can be used to precisely locate the long edges of the runways. The standard HT fails to find runway edges reliably because there is too much distraction from other edge pixels in the image, particularly taxiways and roads. To make it detect runway edges, we must force it to ignore these distractions, by constraining it to only consider edge pixels within a region of interest which is suspected to contain a runway. We form these ROIs by identifying the rough location of each runway in the image, in terms of its central axis line (in parametric $\rho, \theta$ form). To identify these central runway axes, we exploit the only property which is unique to and ubiquitous among all runways, namely, that they contain many runway markings, e.g. centre-line markings (see Figure 2.2). These markings are white, elongated shapes on a dark background, easily detectable and with a well defined direction which is parallel to the runway. They can be extracted by searching the edge map for elongated loops of edges. These objects are found by taking a Canny edge map[10] and removing from it all connected components that do not satisfy certain criteria (see Figure 2.5), in such a way that the only remaining components are elongated loops.

We begin by filtering components with too many or too few pixels. We found that

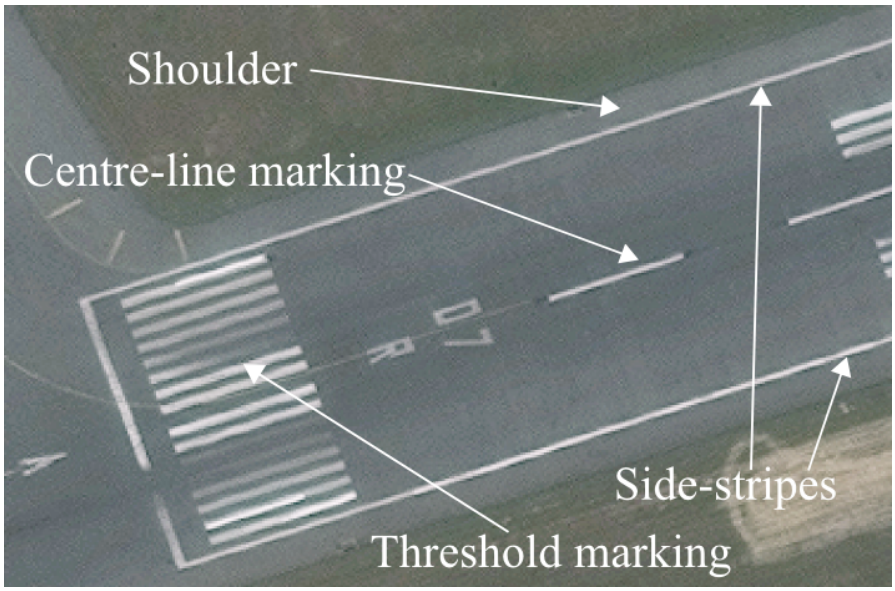
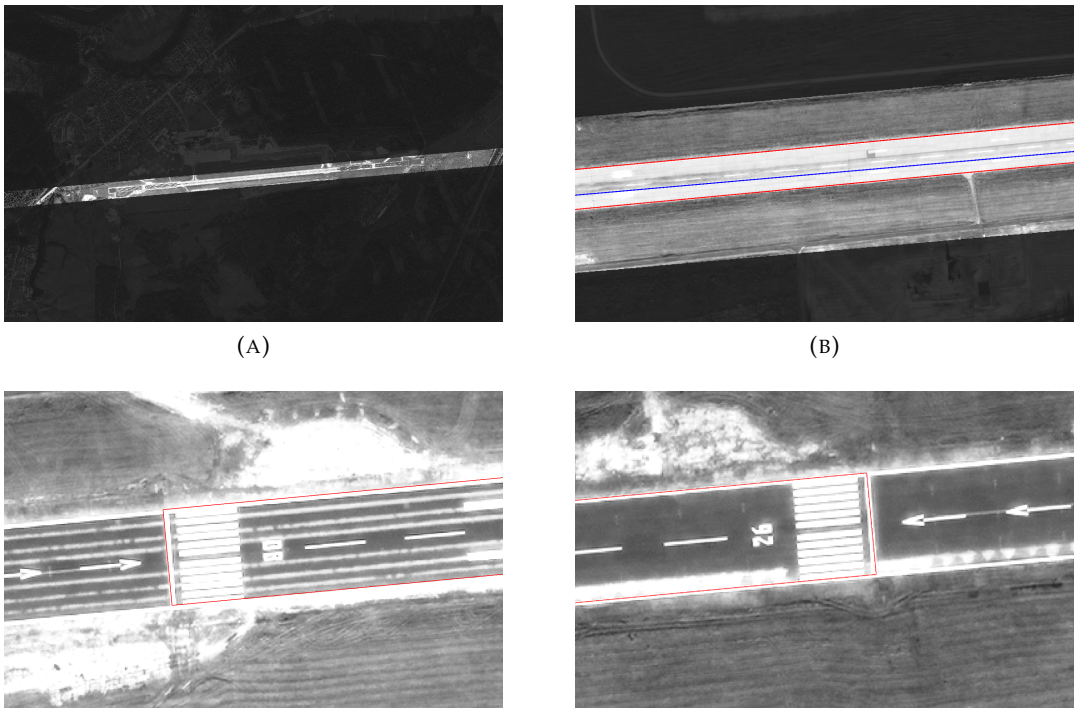



FIGURE 2.2: At high resolutions, more detail becomes visible on a runway. For the purposes of our algorithm, we define the edges of the runway by its side-stripes, rather than its shoulder edges. Our algorithm also utilises threshold markings to delimit the ends of the runway, since they become discernible in VHR images.



(A) (B)

(C) (D)

FIGURE 2.3: Overview of the runway segmentation process. In (a), the region of interest - found by broadening the runway's axis line - is highlighted. In (b), a standard HT finds the long edges of the runway, displayed in red; the axis line is displayed in blue. In (c) and (d), the threshold markings have been found and the runway region capped to produce a rectangular polygon.

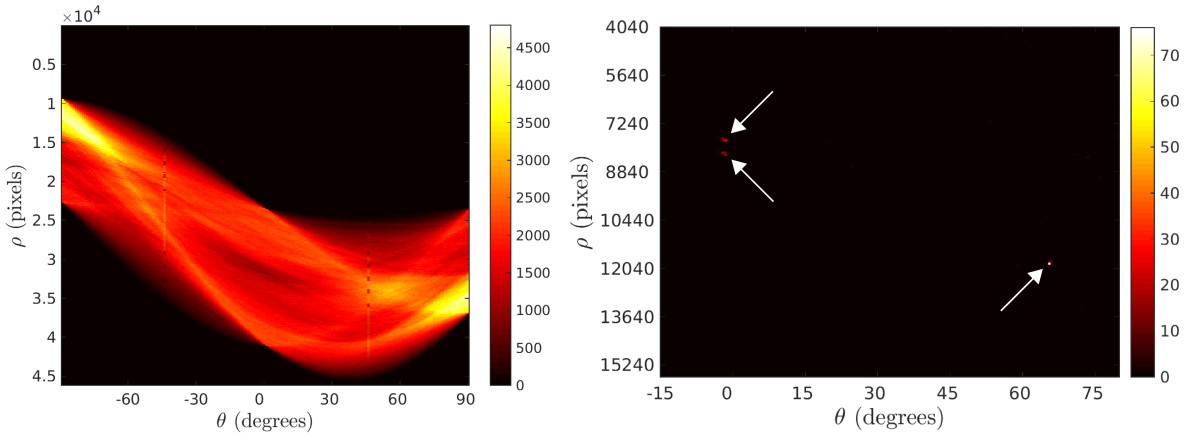
fixed upper and lower bounds were sufficient for our dataset, in which there is negligible variation in both image resolution and the physical size of the markings. We then filter out components with eccentricity below 0.95, i.e. keeping long, thin elements, and finally remove components that are not closed loops. Roughly 20% of the remaining components correspond to runway markings. These connected component parameters are computed using MATLAB's Image Processing Toolbox.

Once the closed loops have been found, they can be used to populate an accumulator array analogous to the edge pixel-based accumulator array as used in the HT. In the standard HT, each edge pixel casts a "vote" towards every line that it could possibly belong to (i.e. each line intersecting that pixel at any angle), by incrementing the value of those lines' accumulator cells. However, since runway markings have both a well defined direction and position, they can each lie along only one line, and hence cast a vote towards exactly one accumulator cell. The parameters of this line are given by,

$$\theta = \left(\phi - \frac{\pi}{2}\right) \mod \frac{\pi}{2} \quad \text{and} \tag{2.1}$$

$$\rho = x\cos\theta + y\sin\theta, \tag{2.2}$$

where $\phi$ is the angle the loop makes with the $x$-axis, and $(x, y)$ are the coordinates of the loop's centroid. In this way, edge loops from the same runway will all increase the same accumulator cell, resulting in one clear peak per runway. As with the standard



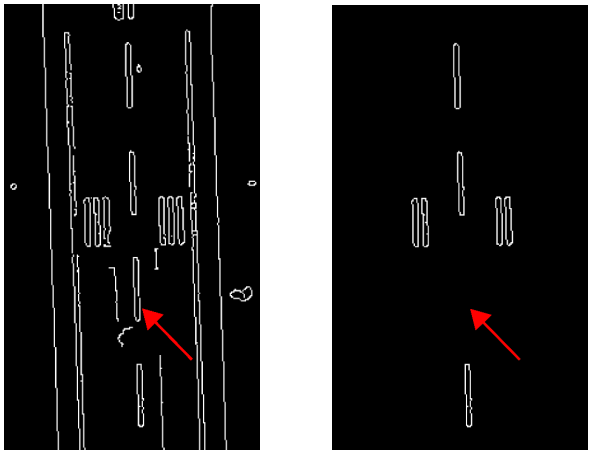

(A) Hough Transform (B) Runway Transform

FIGURE 2.4: Comparison between a standard Hough transform (a) and the "runway transform" (b). The HT does contain many peaks which may be detected (although they are difficult to see with the naked eye), but most of them do not correspond to runway edges, and they are difficult to distinguish from the background noise. The runway transform on the other hand contains three very clearly defined peaks, one per runway (computed from Istanbul International Airport). Figure (b) has been cropped and upscaled for visibility.

HT, the row and column indices of this peak in the accumulator array correspond to the runway's $\rho$ and $\theta$ parameters, respectively. Since the centroids of the markings are estimated with some measure of standard deviation (in particular, individual threshold stripes will have different $\rho$ values since they are spread across the runway rather than along it), we quantise the $\rho$ axis such that they are rounded to the nearest 80 pixels. This ensures that votes from runway markings on the same runway will be concentrated in a narrow range of accumulator cells, and so the peak will be taller than if they had been spread over different cells.

The peaks in "runway space" are very sharp and well defined compared to those in Hough space (see Figure 2.4), since votes from runway edge loops will be clustered at the same location in the transform space, while votes from other edge loops will be spread almost randomly. Also, since each loop casts only one vote, there is far less background noise than in Hough space. Although, as shown in Figure 2.5, some markings are not detected, the runway peaks are still clearly distinguishable due to the low background noise. These peaks are easily detected (and counted), by simply thresholding the transform space at a fixed, empirically chosen value of eight, i.e. at least eight markings must be found along the same line for a runway to be detected.

Once these lines are extracted, a region of interest can be produced for each runway by broadening the line into a strip. Since our dataset has constant resolution and little variation in the physical width of the runways, a constant strip width of 300 pixels was found to work on all images. This parameter only needs to be a rough upper bound on the runway's pixel width, and can easily be calculated if the image resolution and the runway's physical width are approximately known. It is only necessary that both the runway's side-stripes are contained within the ROI. The standard HT is then applied within this region to locate the long edges (see Figure 2.3). By restricting the HT to consider only edge pixels from within the ROI, we remove the majority of the non-runway edge pixels and make the runway edges appear far more prominent in the



(A) (B)

FIGURE 2.5: A runway region from the Canny edge map, showing (a) original edges and (b) elongated loops left after connected component filtering. The arrow indicates a marking which was not detected, due to a small break in the edge loop.

Hough space. This also reduces running time for the HT, which is linear in the number of edge pixels.

Since we define the boundary of the runway as the white side-stripes rather than, for example, the outer edges of its shoulders (see Figure 2.2), we use the result of a local mean threshold as the input to the HT, rather than an edge map. The local mean threshold responds well to the white objects on a dark background, and will only produce a single line for each side-stripe, unlike an edge transform which would double lines (one either side of each stripe). We use a circular window of 5 pixel radius for the local mean threshold. In this way we ensure the HT detects specifically the white side-stripes, rather than any other long edges.

Due to the length of runway edges in VHR data (at least 3000 pixels in our dataset), an angular resolution of one sixteenth of a degree is used in the HT to allow the lines to fit closely along the full length of the runway.

Having extracted the runway's long edges, it is still necessary to find the threshold markings that terminate the runway (see Figure 2.6a). These markings are distinctive in that they are periodic: normally 4-16 (depending on runway width) [11] white stripes of even spacing and thickness. Our algorithm identifies these markings by first rotating the image so the runway lies along the $x$-axis, then cropping so only the strip between the Hough lines remains. To provide some robustness against small breaks in the threshold stripes, we perform a morphological greyscale closing, using a horizontal line structuring element. The algorithm then analyses independently all columns of pixels from the strip, each of which is a cross-section through the runway. Columns are classified as threshold or non-threshold; the classification requires three criteria to be met:

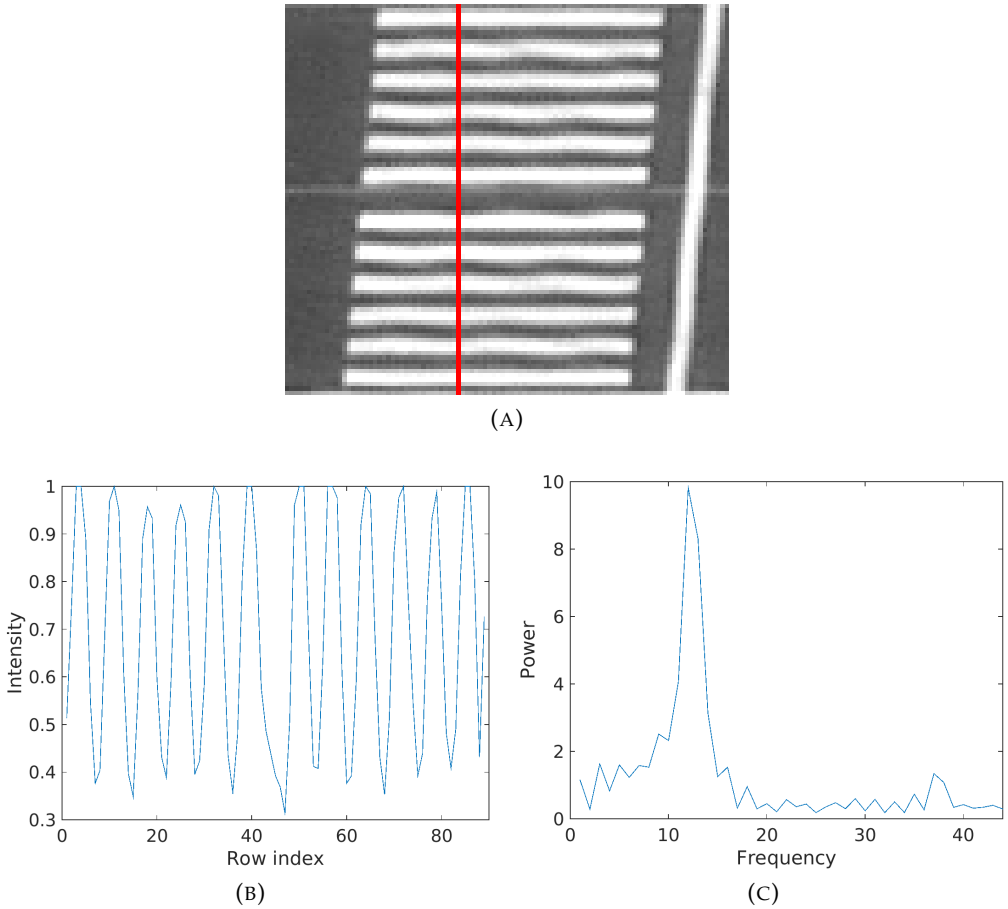
- Brightness: thresholds are marked in white paint, therefore a column that runs across one must have a greater mean intensity than that of the whole strip.
- Number of stripes: threshold markings consist of 4-16 white stripes. In a single column, this appears as 4-16 intensity peaks (see Figure 2.6b). We count these by performing a 1-D local mean threshold (with window size equal to a quarter

of the column length) within the column, and counting the number of connected components that appear. Only columns with 4-16 stripes are classified as threshold; this allows for some noise and/or blurring of stripes.

- Periodicity: threshold markings are stripes that repeat at a regular frequency (see Figure 2.2). As such, a clear frequency peak should be found in the column's Fourier transform (see Figure 2.6c). We consider such a peak to be present if the maximum Fourier coefficient is at least 1.5 times the mean Fourier coefficient.

Once the columns have been classified, the threshold markings will appear as large contiguous groups of threshold marked columns, the outer edges of which mark the boundaries of the runway.

The whole process consists of many steps, and many of those steps involve parameters which have been found empirically. However, most of these parameters are dependent on the dimensions of various runway features in the input images, and should work across a whole dataset (as they do with ours), so long as those dimensions are roughly consistent. In addition, this long series of tests ensures that no false positives are caused by other long, straight objects such as roads or taxiways. For example, even though roads are long, dark and have markings similar to runways, they still fail at multiple stages, since they have no side-stripes or threshold markings, and




(A) (B) (C)

FIGURE 2.6: (a) Threshold marking taken from a runway strip. (b) Intensity profile of the highlighted column in (a). (c) 1D Fourier transform of the intensity profile.

their centre-markings have too few pixels to survive the connected component filtering step.

## 2.4 Results

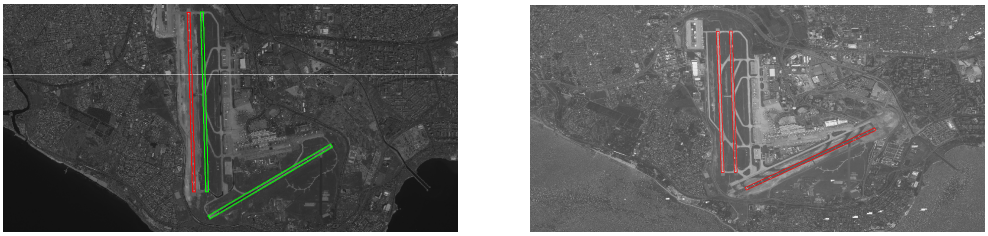
We tested the algorithm on six bi-temporal pairs of images from the QuickBird II satellite, with a panchromatic resolution of 61 cm. These images vary in size from around 25-250 megapixels. Figure 2.7 presents the algorithm's output on three of six airports from the input dataset, provided by Jeppesen. It can be seen how in some cases the airport itself comprises only a small proportion of the total original image area. We evaluate the accuracy of the algorithm by comparing the segmented runway polygons with those from Jeppesen's ground-truth GIS data. To quantify how similar these polygons are we compute Jaccard indices [12], given by

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{2.3}$$

for two polygons A and B, where $A$ is the set of pixels belonging to polygon A, and $B$ the set of pixels belonging to polygon B. Table 2.1 shows the detection rates and mean Jaccard indices for the detected runways for all six airports.

The algorithm took approximately four minutes to terminate on a quad-core Intel Core i7-4790 CPU at 3.6GHz, processing our largest image at 256 megapixels, which contains three runways. Our implementation was written in MATLAB.

Between all twelve images, there are 26 runways, all but two of which are detected by the algorithm. The two failure cases were caused by heavy erosion of those runways' threshold markings - if they are too faint due to weathering then they will not be conclusively detected, and the runway's existence will not be reported. This issue is clearly shown in Figure 2.8, the sixth airport, where the runway markings are dulled by apparent tire marks and jet exhaust. These issues might be partially mitigated by using some pre-processing, such as histogram equalisation.

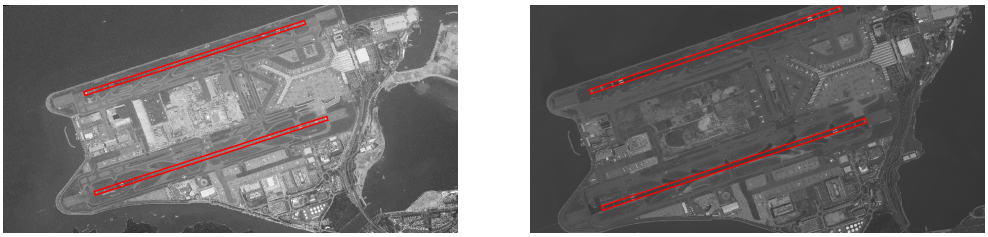


(A) Istanbul (May 2006)

(B) Istanbul (June 2011)

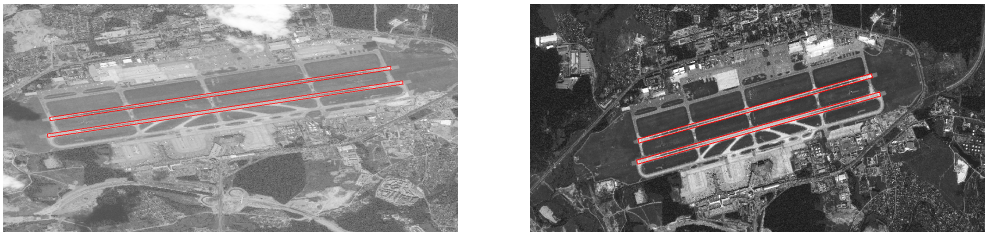
(C) Bahrain (May 2011)

(D) Bahrain (April 2014)



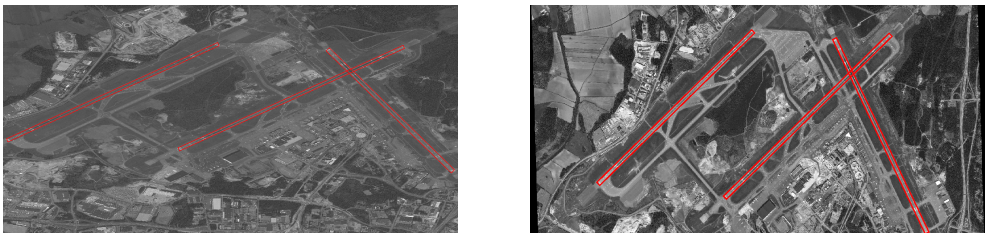
(E) Hong Kong (June 2008)

(F) Hong Kong (April 2014)



(G) Moscow (August 2011)

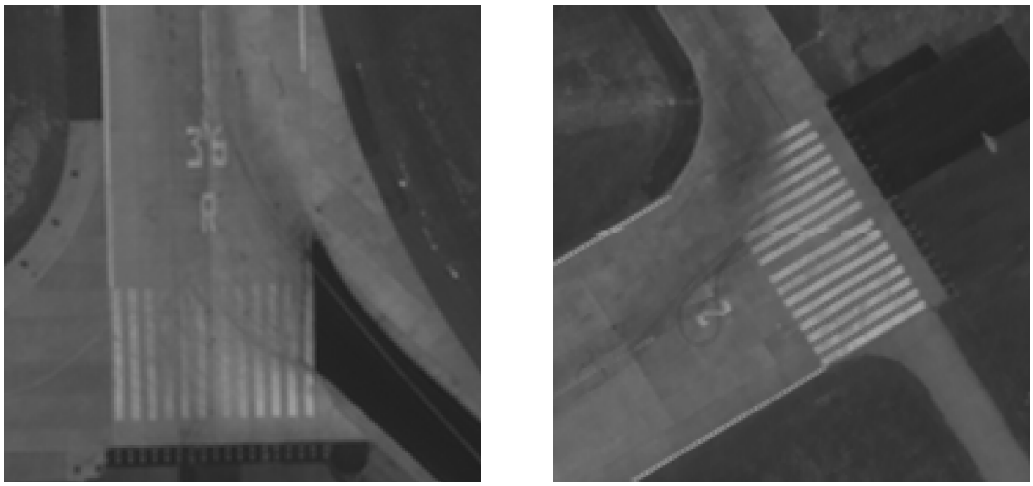
(H) Moscow (July 2014)



(I) Helsinki (May 2008)

(J) Helsinki (August 2014)

FIGURE 2.7: The algorithm's output for six representative images from of our sample dataset, provided by Jeppesen. These images have been cropped to focus on the runway region, which can be a small proportion of the original images. The green boxes show the ground truth (from pre-existing GIS data) for the two runways which were not detected.



(A) (B)

FIGURE 2.8: Heavily eroded threshold markings caused by taxiing planes lead to the runways remaining undetected.

TABLE 2.1: Jaccard indices, comparing detected runway polygons with those marked by human experts.

| Airport | Runways Detected | Jaccard Index |
|---|---|---|
| Kaunas (2008) | 1 of 1 | 0.978 |
| Kaunas (2009) | 1 of 1 | 0.980 |
| Bahrain (2011) | 2 of 2 | 0.976 |
| Bahrain (2014) | 2 of 2 | 0.969 |
| Hong Kong (2008) | 2 of 2 | 0.974 |
| Hong Kong (2014) | 2 of 2 | 0.978 |
| Moscow (2011) | 2 of 2 | 0.969 |
| Moscow (2014) | 2 of 2 | 0.970 |
| Helsinki (2008) | 3 of 3 | 0.967 |
| Helsinki (2014) | 3 of 3 | 0.974 |
| Istanbul (2006) | 1 of 3 | 0.954 |
| Istanbul (2011) | 3 of 3 | 0.968 |

## 2.5 Conclusion

In this chapter we have presented an algorithm for the precise detection of runways from VHR remote sensing data. We find regions of interest by observing runway markings that are not discernible in lower resolution imagery, and produce vector polygons that precisely fit the runway area. Accurate results with a zero false-positive rate are

demonstrated in our testing dataset. The algorithm is robust against variations in runway intensity and variations in outline shape caused by adjoining taxiways and other artefacts. Its performance is also unaffected by the presence of large tire marks down the runway centre, even when they obscure the majority of the runway markings. It can detect a variable number of runways, and does not return false-positives by confusing runways with taxiways or other background features such as roads.

Since the images we have worked with are geo-referenced, the runway polygons produced by this algorithm can be trivially converted into physical longitude/latitude coordinates, allowing for easy comparison with existing runway shapefiles (indeed this is how we obtained the Jaccard indices in Table 2.1). Automated detection of changes to runway area is therefore a trivial extension of this work. Also, although this chapter only deals with segmenting the entire runway area as a single polygon, future work may draw on this to perform more detailed change detection. The many runway markings that our algorithm relies on are themselves AMDB features, and furthermore they are small, numerous and likely to change often due to repainting, and therefore may be particularly tedious for the AMDB team to map by hand. This, along with their morphological simplicity (white rectangles on a dark background) would make them a good target for automatic detection.

In addition, a number of performance improvements can be made. For instance, although finding edge loops has proven the most effective way of detecting runway markings so far, it is a lengthy process, and could potentially be replaced by a local thresholding operation, a selection of which are listed in chapter eight of [13]. The other large performance bottleneck was rotating the entire airport image in order to align a runway with the $x$-axis. A future version of this algorithm which detected threshold markings without needing to rotate the entire image and perform column-wise analysis would execute much faster.

# Chapter 3

# Registration

## 3.1 Introduction

One of the main deliverables of this project is automatic change detection between bi-temporal image pairs. If no change has occurred to an airport then no change need be made to the AMDB, therefore change detection has a high priority. However, since the two images are generally captured with the satellite in different positions in the sky, the two images must first be co-registered before any direct comparison can take place. Although image registration can generally be accomplished by manually placing control point pairs at matching locations in the two images, from which a geometric transform can be computed and applied, we decided to briefly research the feasibility of fully automatic image registration. This section details the techniques we investigated to solve this problem.

## 3.2 Background

The most common approach to automatic registration assumes that since the two images are of the same scene but taken from different positions or angles, then at least some of the same features should be present in both, but at different positions on the image plane. If enough of these features can be detected in both images and matched together, then a geometric transform can be estimated, and applied to one image to line it up with the other. These feature pairs can always be picked out and matched manually, but it is possible to both detect and match the features automatically. There are a number of algorithms for detecting "feature points", such as SIFT (Scale Invariant Feature Transform) [14] and its high-speed descendant, SURF (Speeded-Up Robust Features) [15]. SURF and SIFT both work by detecting blob-like features by finding local maxima of the determinant of the Hessian matrix, and also describe those features by assigning them each a numerical descriptor vector. These descriptors aim to be invariant to the scale, orientation, illumination and (to some extent) affine distortion of the features, allowing corresponding features in different images to be robustly matched even when these parameters differ due to (in the case of remote sensing data) satellite position and atmospheric conditions. Other feature point algorithms include FAST (Features from Accelerated Segment Test) [16] and Harris features [17], which detect and describe corners, and BRISK (Binary Robust Invariant Scalable Keypoints) [18], which is even more computationally efficient than SURF, and describes feature points with bit-strings. Image registration is a vital precursor to change detection in remote sensing data - in fact, it has been demonstrated that sub-pixel registration accuracy is required for the accurate performance of many change detection algorithms [19]. As such, the literature contains several papers which advance new techniques for this purpose. Intensity based methods use similarity functions between two images,

such as normalised cross-correlation (NCC) and mean square difference (MSD) [20] as similarity measures between two images, and register the images by finding a transform which maximises this similarity. Chen *et al.* [21] demonstrate weaknesses in these two similarity measures and propose a new one based on mutual information, a statistical method for comparing two probability distributions [22] (which in this case are the images' normalised histograms). Li *et al.* note that incorrect feature point matching is common when SIFT is applied to remote sensing data, and claim to reduce these false matchings by introducing a Joint Distance metric, which depends not only on the Euclidean distance of two descriptor vectors but also their orientation and scale mismatch.

## 3.3 Methods

### 3.3.1 SURF point matching

MATLAB has extensive built-in functionality for finding many types of control points and extracting their descriptors, so we began by attempting to register our image pairs by SURF point matching, as demonstrated by ([23], [24]). However these cases both found control points in much smaller regions than our images, and when we tried the technique on our images, we obtained almost no correct matches (see Figure 3.1). Other feature point detectors yielded similar results.

It seemed that the low matching rate was probably caused by the large size of our images, especially since the methods used in [23] [24] limit their search space for feature points to small image regions. Automatically discovered control points were difficult to match because so many potential control points are present in VHR images. In practice we must choose a finite number of control points in both images, found by choosing the $n$ strongest features according to the SURF metric ($n = 100$ in Figure 3.1). Since $n$ is normally a small fraction of the total number of strong features, there is little overlap between these sets - so a control point in the old image often has no match found in the new. This can be partly alleviated by increasing $n$, but then it becomes harder to match the control points reliably, since they become less unique the more of them there are. The problem was worsened further by the changes that take place in the scene during the years that separate the two images; this often meant that a feature found in the old image was no longer present in the new. This was especially true for mobile objects like cars and boats which SURF often detected.

### 3.3.2 Runway corner matching

Since matching SURF features and other automatically discovered control points proved to be so difficult in such large images, we instead tried to use the corners of the airports' runways as control points. These corners can be either extracted by the runway detection method in the previous section, or from pre-existing GIS data. By using runway corners, we generate control points which should (unless runways are added, removed or changed between the two images) at least have a valid matching between the two images.

For the old image, we can assume that GIS data which demarcates the runways already exists, from which the runway corners can easily be extracted. However in the new image, we assume that no GIS data exists, therefore we use the runway detection algorithm in the previous section to detect the runways in the new image. The matching step itself is the same as with standard feature point based registration; we

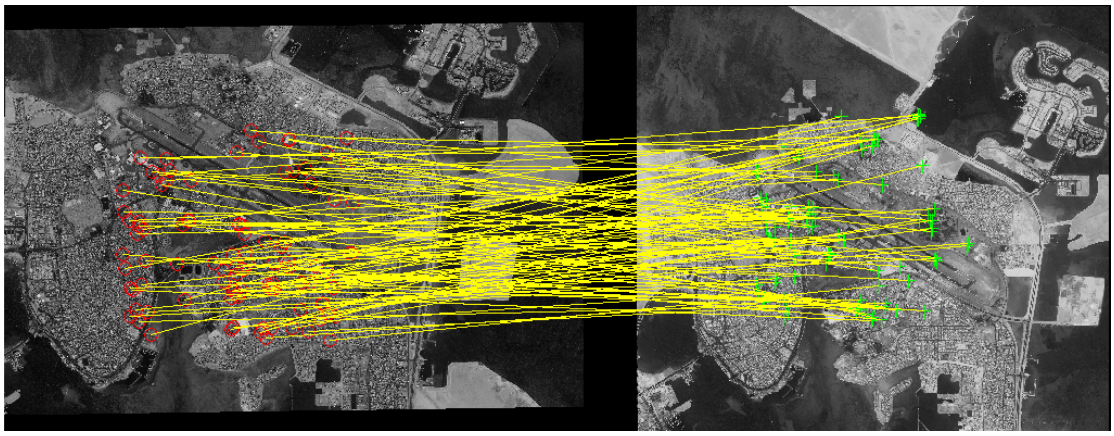


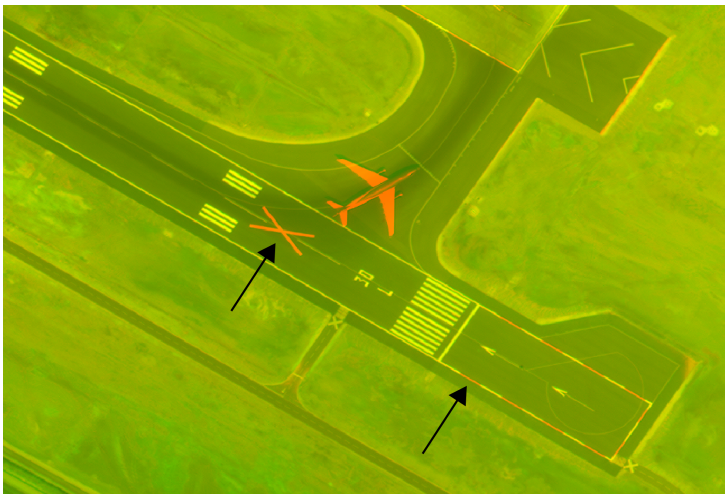
FIGURE 3.1: Applying SURF point generation and matching to the Bahrain bi-temporal image pair produces almost no correct matches. In this example we limited SURF point detection to the center of the images to reduce the number of points found near the border, which may have no correct match due to image translation. We also tuned the parameters to detect large features ( 100x100 pixels), in the hope that they might be rarer and therefore easier to match than small ones.

compute a descriptor vector for each control point and match them by Euclidean distance. We obtained best results using the FREAK (Fast Retina Keypoint) descriptor, obtaining around 75% correct matching rate in most cases. This is sufficient to estimate the affine transformation needed to register the images. To this end we use the `estimateGeometricTransform` function from MATLAB's Computer Vision System Toolbox, which uses RANSAC to detect the inlier matches which conform to one consistent pattern, and discard outlier matches which do not fit this model. The incorrect matches are caused by the fact that our algorithm produces slightly different runway polygons to the ones human experts produce, particularly when the satellite captures the airport from an oblique angle, causing skew distortion on the runways (our algorithm always outputs a rectangle even when the correct polygon should be a parallelogram). This difference in control point positions sometimes causes significant registration error when this method is used. This error varies from one or two pixels in cases where the satellite captured the new airport image from overhead (in which case our algorithm detects runway corners almost identically to a human operator) to 15 pixels or more when significant skew distortion is present. The error could possibly be reduced by using a different pattern of control points from the runways, for example four equally spaced points along the polygon's central axis instead of the polygon's corners. However, a set of co-linear points as described would not provide sufficient information to generate an affine transformation if only one runway was present in each image.

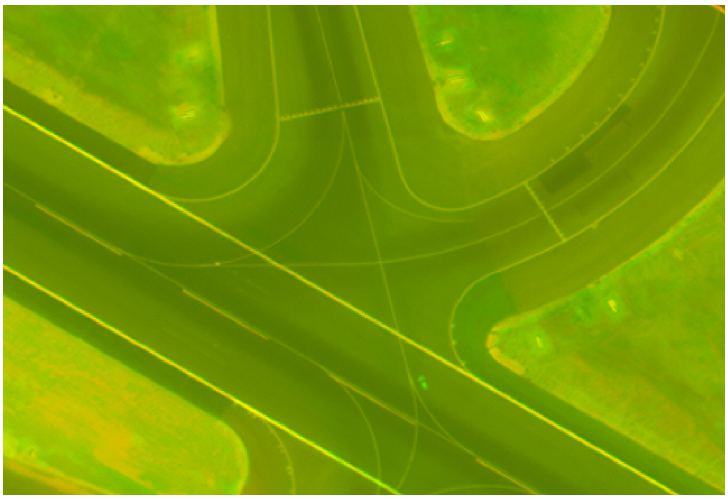
Far better results ($< 1$ pixel registration error) are obtained when human-annotated GIS polygons are used to extract both sets of runway corners (see Figure 3.2). However, this method is very much akin to registration by manual control-point selection, therefore we do not consider this as an automated registration method.

### 3.3.3 Using coarse registration to improve SURF point matching

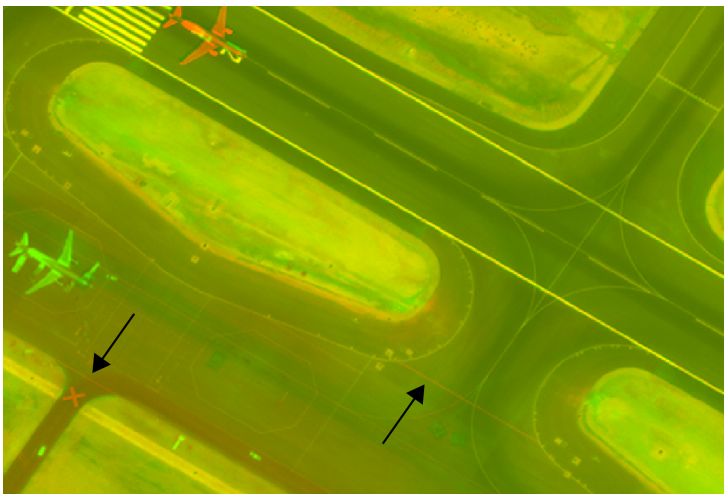
Matching of runway corners often produced only a coarse level of image registration with significant error, due to differences in the polygons extracted by our algorithm and



(A) Runway markings added



(B) No change



(C) Stopbars added (bottom right and bottom left)

FIGURE 3.2: Registration results when using corners of manually annotated runway polygons on both the old and the new image. The red channel displays the new image, while the green displays the co-registered old image. The lack of red or green borders around yellow shapes shows a high degree of accuracy, furthermore, red shapes can been seen in some places indicating runway or taxiway markings that have been added.

by human operators. However, we were able to use this coarse preliminary registration to assist in the matching or SURF features, hoping to then produce a more accurate registration. SURF point matching was tried originally, but suffered from low correct matching rate (Figure 3.1). By performing a coarse preliminary registration we can hugely narrow down the search space when searching for SURF point matches, because for a given point in one image (the location of a detected SURF point), we now know that its true corresponding position in the other image must be somewhere in a small window centred on the same pixel location. The size of this window depends on the registration error, for which we took 15 pixels as an empirically found upper bound, giving a square window of 31 pixel width.

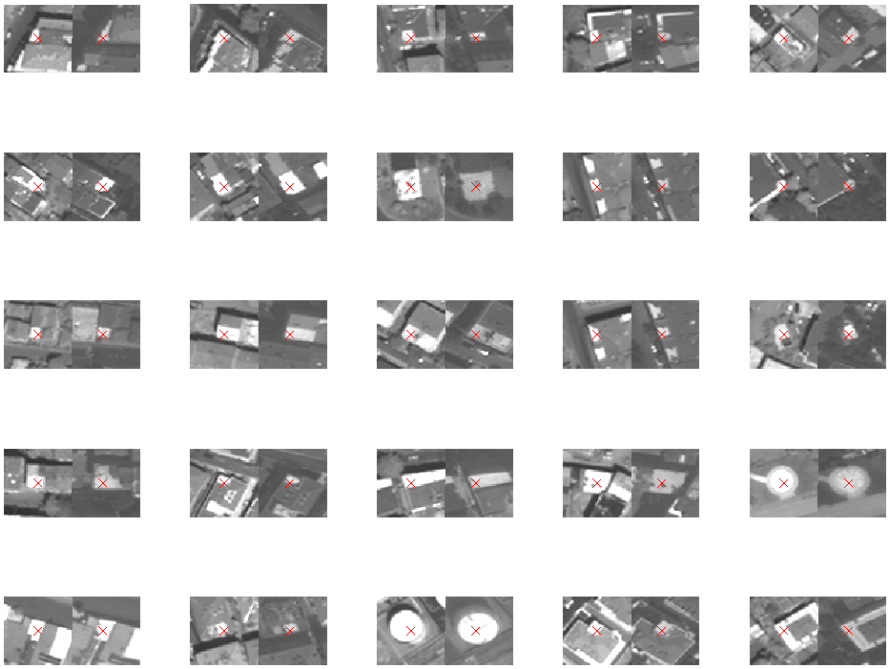
This method succeeds in greatly improving the correct matching rate as demonstrated with the Istanbul International Airport image pair (see Figure 3.3(a). However, despite the high number of correct matches, significant registration error is still found after the affine transform is estimated and applied (see Figure 3.3(b). The reason for this is unclear, although we hypothesise that it is due to most of the control points detected being on the roofs of buildings. Since one image in the pair was captured from directly overhead while the other was from an oblique angle, this may result in roofs of buildings being correctly aligned while the ground plane is mis-aligned.

### 3.3.4 Registration by georeference data

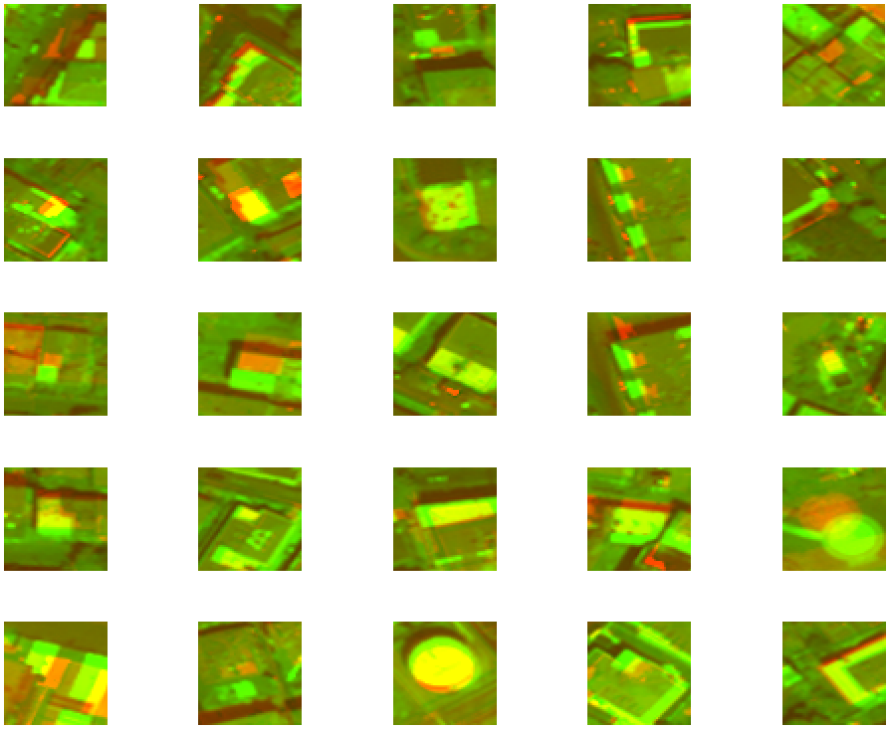
Finally, we found that it was possible to register images using their embedded georeference data, which we extract from the image files using the `geotiffread` function from MATLAB's Mapping Toolbox. With it, we are able to achieve a level of accuracy comparable with that attained by matching GIS runway corners with detected runway corners. This georeference data effectively describes for both images a rectangular region on the Earth's surface in longitude and latitude coordinates, hence mapping every pixel to a geographic coordinate pair. The assumption that the part of the Earth's surface captured in a satellite image is rectangular (both in the Cartesian coordinate frame and the polar longitude/latitude frame) is technically false, but it is a good approximation, for two reasons:

a) The portion of the Earth's surface covered by the image is small enough relative to the Earth's circumference that the Earth's curvature is negligible, hence we can consider it to be flat, and so the longitude/latitude coordinates of points on that surface are linearly correlated with their x/y coordinates
b) Although the region viewed by the sensor is technically trapezoidal unless the sensor is directly overhead, pointing normal to the surface, we note that the sensor is in low Earth orbit - therefore it is far enough away (relative to the size of the region viewed) that the lengths covered by the top and bottom edges of the image are roughly equal, and the sides are parallel.

This allows us to determine the geographic region of intersection and extract the corresponding pixel regions from the two images, scaling them as necessary in the $x$ and $y$ directions to overlay correctly. Since our images are already rotationally aligned, this is theoretically sufficient to produce accurate registration. However, we have found that the georeference data for two images in a bi-temporal pair will actually predict slightly different longitude/latitude values for the same feature in the two images, and is therefore not perfectly accurate. Hence, this process results in a registration error of around 10 pixels on average. This method can be performed completely automatically as long as georeference data exists, and theoretically could be made arbitrarily

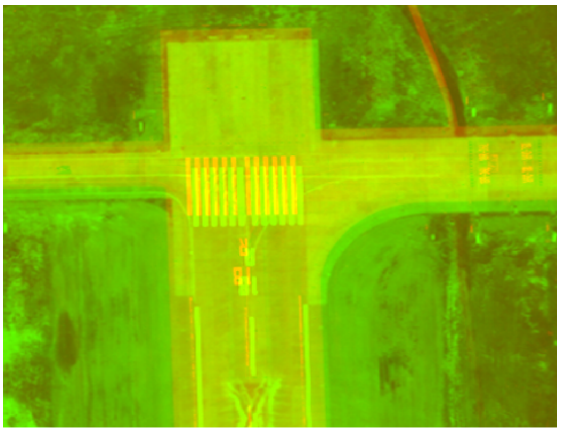


(A) Matched SURF point pairs

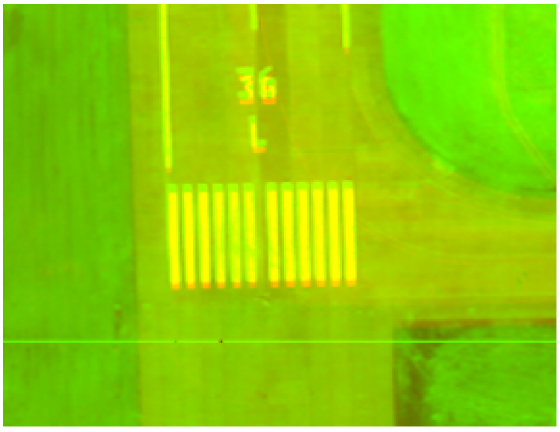


(B) Comparison of above features post-registration

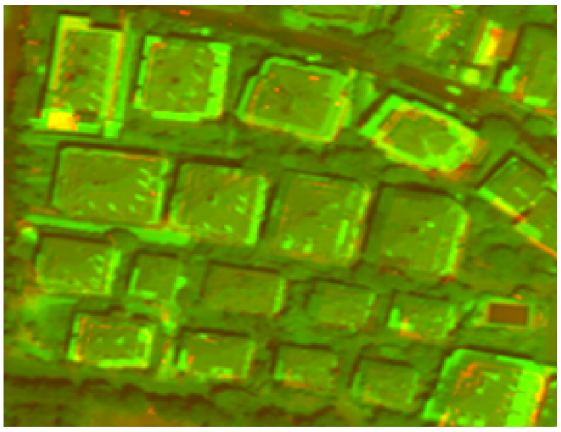
FIGURE 3.3: (a) Matched feature pairs, with the feature in the old image on the left and the match found in the new image on the right (red crosses show the precise locations of the SURF points). It is noteworthy that most of these features are on the roofs of buildings. (b) Significant and non-systematic mis-alignment is present post-registration, even though (a) shows that most of the features have been matched correctly.



(A) Runway north end - significant registration error




(B) Runway south end - registration more accurate



(C) Urban residential area

FIGURE 3.4: Comparison of three regions of Istanbul airport after registration using runway corners, followed by SURF point matching.

accurate by improving the accuracy of the georeference data. We hypothesise that the inaccuracy in the georeference data is caused by the assumption that the image spans a rectangular region on the ground. A registration error of 10 pixels, although very significant, is also very small relative to the size of the image, therefore the rectangular assumption need only be slightly inaccurate in order to account for it. It may be possible to produce a more accurate georeference model if one has access to the satellite's position and orientation data at the time of image capture.

## 3.4 Conclusion

Image registration is a vital precursor to most forms of change detection, because having the same geographic position occur at the same pixel location in both images allows element-wise operations to compare that location between the two images. In this section we tested several methods for automatic or semi-automatic image registration. We found that VHR data presents difficulties for registration which are not present in lower resolution data. For example, the QuickBird II sensor is powerful enough to resolve small objects such as vehicles and other non-static objects, which do not occur in the same place in both images. Therefore they are detected as feature points but cannot be matched together. We found that SURF point matching in such large and detailed images was difficult, since the images present so many potential feature points (blob and corner-like objects) that we could not ensure that a) the same objects would be detected in both images (since the number of objects detected must be capped at a finite number), and b) that correct matches would be unambiguous, since there are often many similar-looking features. We also found that most of the detected feature points were on the roofs of buildings, which presented problems when the images were taken from different angles because even if the rooftop features were correctly lined up, the ground plane may still be mis-aligned. Intensity based methods are also quite ineffective in VHR data, because of the many small-scale changes which are visible.

In summary, we found that the most effective registration method was to extract control points from manually annotated GIS data. This method is manual in the sense that GIS polygons are created by hand, although the matching and transformation estimation steps are automatic. Given the accuracy of the results this achieved, it maybe that those results are worth the cost of manually registering the images if no fully automatic process can be found, especially seeing as eight control points per image was found to be sufficient. Doing so allows the production of images such as those in Figure 3.2, which may be used to greatly speed up the change detection process, either by making changes obvious to a human operator or by being processed further by another algorithm. The most effective fully automatic result we attained was by matching runway corners from GIS data in the old image with detected runway corners in the new image. By choosing control points in this way we guarantee that valid matches will exist (so long as the runways are the same in both images), and that the control points themselves lie on the ground plane. This technique can likely be improved by searching for more ground-plane control points, such as taxiline intersections. This will improve robustness against cases where the runways do change, or when the runway corners are detected differently in the two images. Future research should aim to implement techniques from the literature which are specifically developed for VHR data, since methods developed for lower resolution images were found to be ineffective on our dataset.

# Chapter 4

# Change Detection

## 4.1 Introduction

One way in which automation can speed up the change detection process is by detecting spatial regions within a bi-temporal image pair where meaningful change is likely to have occurred. This information can then direct the attention of a human operator (or another automated process) to such a region. Furthermore, algorithms found to perform well at detecting large scale changes are likely to also detect changes in smaller scale features such as runway or taxiway markings when run on full resolution data, if sub-pixel accuracy registration of full scale images can be achieved.

The challenge in this chapter is to recognise when a meaningful change in shape and structure has occurred. With change detection, there is potential for false positives to be caused by factors such as changes in contrast, shadowing due to sun position and specular highlights. It is also unavoidable that some irrelevant changes will be picked up, since a given image pair may contain many interesting changes that lie outside of the airport. Even within an airport, resurfacing on runways and taxiways and the movement of planes are examples of changes that any low level algorithm would struggle to distinguish as unimportant. In general, a large amount of change does occur between airports at different dates, and classifying those changes as relevant or irrelevant is somewhat subjective.

Change detection is a large field, with applications in areas as diverse as video surveillance, remote sensing, medical diagnosis, civil infrastructure and underwater sensing. As such, there are many change detection algorithms specialised to different purposes. In this chapter we test a variety of change detection techniques from the literature, along with several novel techniques. We find that one of our novel techniques, called Local Ternary Patterns, produces particularly useful results. Many of the approaches tested in this chapter are covered in a literature review by Radke *et al.* [25], who kindly provided MATLAB code samples implementing some of those methods.

## 4.2 Background

A more recent literature review by Hussain *et al.* [26] covers a variety of change detection algorithms specific to remote sensing imagery, and classifies them as either pixel based or object based. The pixel-based methods involve taking two co-registered images and considering pairs of corresponding pixels in isolation (i.e. without considering the pixels' spatial context), and comparing the spectral values of those pixels do decide if a change has occurred. Pixel based methods are most suited to applications where the physical area covered by a single pixel is very large (for example as with images from the Landsat satellites), that is, large enough to fully contain the objects being searched for. These methods either directly compare the spectral channels of the

pixels (e.g. red, green, blue, infra-red), or metrics derived from those values, such as the tasselled cap transform which computes brightness, greenness and wetness values for each pixel. For example, Nordberg *et al.* [27] use tasselled cap in conjunction with change vector analysis and principal component analysis to decide which pixels show deforestation. There are also post-classification methods ([28], [29]) which classify the pixels of each image first, then detect which pixels change class. These methods bypass the need to correct for illumination or atmospheric differences, since the images are classified separately, but can suffer from poor accuracy since they require both images to be classified correctly. Since both prior classifications must be correct for the comparison to be valid, the accuracies of the two classifications are effectively multiplied, giving a lower final accuracy (e.g. $80\% \times 80\% = 64\%$) [30].

Pixel based change detection algorithms are well studied and have a long history of use, but are generally inappropriate for VHR data such as ours, since we are concerned mostly with the appearance or disappearance of objects comprised of many pixels. It is difficult, for example, to reliably conclude that a pixel has transitioned from dirt to taxiway by considering only its RGB values, since taxiways do not have unique and consistent RGB values in general. Hussain's review [26] also describes "Object based" approaches, which detect objects in both images then attempt to match them based on properties such as texture, shape, colour (or other spectral values or derived metrics) and spatial relationships with neighbouring objects. Using the runway detection algorithm in Chapter 2 to detect new or extended runways by comparing the runways detected in two bi-temporal images would be an example of object-based change detection. Another example is given by Miller *et al.* [31], who demonstrate a general purpose object-based change detector which is robust against extreme illumination changes. We limit this chapter to consider only pixel or local-region based change detection, since true object-based change detection requires one to first detect objects - a difficult problem that we cover in other chapters.

## 4.3 Methods

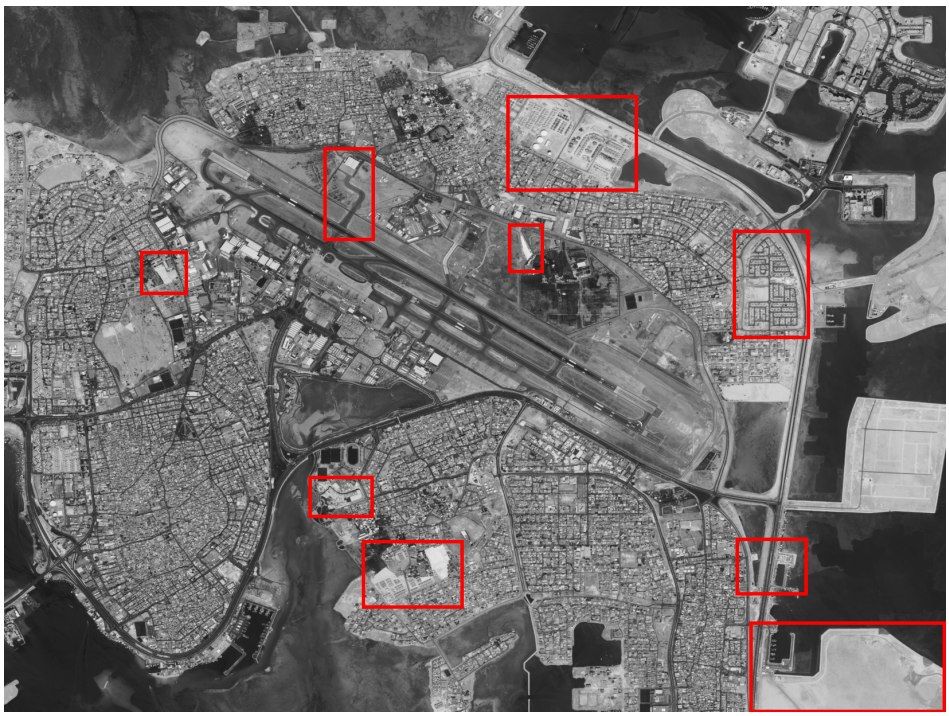
In this report we demonstrate the strengths and weaknesses of a number of simple change detection approaches which we have implemented, tested and validated. For each algorithm we provide a plain-English description along with a demonstration on a test case. All of these algorithms take two co-registered greyscale images as input and produce a difference function as output, defined for all pixels in the image. Where appropriate, we provide a mathematical description of this function. In such cases, we use $I_{old}(x, y)$ and $I_{new}(x, y)$ to refer to the old and new greyscale intensity images (downsized by a factor of eight), and $D(x, y)$ for the difference function being computed. In all cases, histogram equalisation is first applied to the image pairs in an attempt to correct for illumination and atmospheric differences.

To demonstrate the effectiveness of each approach, we use Bahrain International Airport (OBBI) as a test case. Figure 4.1 are two co-registered, down-sized, greyscale images of OBBI, with histogram equalisation applied. Red boxes highlight the regions of the scene in which the most obvious changes occur. Not all of these changes are relevant to the AMDB, but an adequate algorithm should detect all of them nonetheless, since similar changes could occur to relevant AMDB structures. To assess the performance of the algorithms, their difference function outputs are displayed as heat maps with the region of interest boxes drawn in the same places as on the original images.

Capable algorithms should exhibit a strong response within these boxes, and a generally lower response elsewhere. We are particularly interested in finding an algorithm which produces a strong response to the new taxiway and apron added to the north of the airport, since they are a perfect example of the kind of AMDB feature we hope to detect.

For the purposes of these tests we register the images using the embedded georeference data as discussed in the previous chapter, since this is the simplest fully automatic method we found. To deal with the registration errors caused by inaccuracy of the georeference data, we downscale the images by a factor of eight. This leaves us with enough information still to detect large regions of change (such as the addition of a taxiway or a construction site, and improves the running times of the operations greatly, allowing for faster testing of change detection algorithms.

(A) Bahrain (2011)



(B) Bahrain (2014)

FIGURE 4.1: A co-registered bi-temporal image pair of Bahrain International Airport. Red boxes have been drawn in the same locations on each image to highlight areas where meaningful change has occurred. Although these areas are chosen subjectively, it is clear that an effective change detection algorithm should should detect more change in these regions than in most other areas of the image.

### 4.3.1 Simple differencing

$$D(x, y) = |I_{new}(x, y) - I_{old}(x, y)| \tag{4.1}$$

This is the simplest and most obvious approach to direct image comparison, and as such serves as a good baseline against which to compare other techniques. Aside from that however, this method has limited practical use. Although good at detecting areas where the scene becomes consistently brighter or darker, this approach was found to suffer from a number of problems:

- Since each pixel is considered in isolation, the difference map is not necessarily smooth - e.g. if buildings are constructed on a field, then an "imprint" of the buildings will be visible in the change map, rather than an area of consistently high change. This makes it difficult to produce a binary change mask by thresholding $D(x, y)$, since many small regions of change would be detected.
- The method is sensitive to changes in intensity or contrast, in the absence of changes in shape or structure. For example, if a building's roof looks brighter due to changes in sun position producing specular reflection, this will be detected as a change - despite the shape of the building remaining constant.
- Local response is proportional to change in intensity, which in general is only vaguely related to the amount of meaningful change. A low contrast area may change significantly but with little intensity change, producing a low response, while another area may increase in intensity due to re-surfacing or illumination changes and be detected strongly.

Although this algorithm successfully detects the addition of several bright structures, it responds poorly to the new taxiway, since the taxiway has a largely similar intensity to the terrain that was there previously. Generally, this algorithm only produces a solid, noticeable response when there is a large and consistent change in absolute intensity across a region.

This approach did seem to perform better on data that was downsized further - by a factor of 32 instead of 8. The output of the simple differencing algorithm is shown in Figure 4.2.

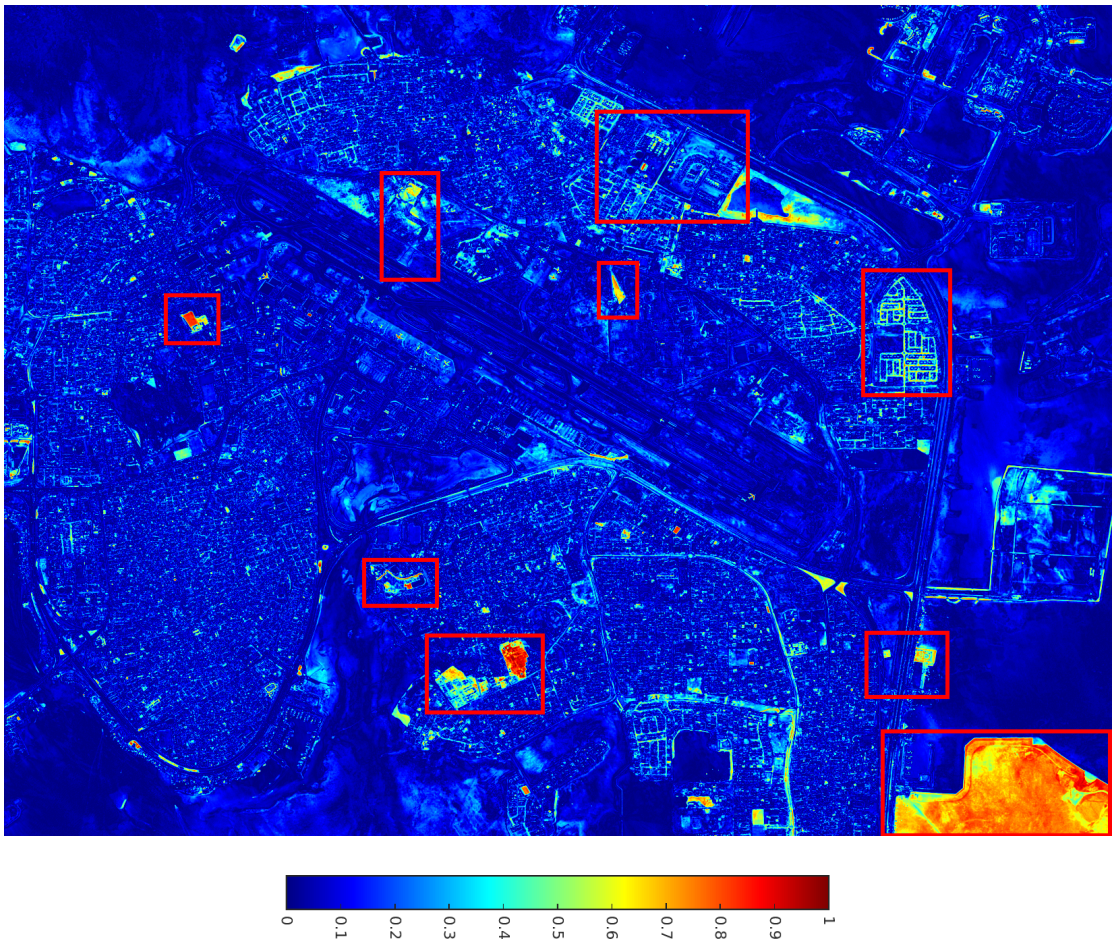



FIGURE 4.2: Change detection heat-map for simple differencing. The colour bar used here also applies to subsequent figures in this chapter.

### 4.3.2 Image ratioing

$$D(x,y) = 1 - min\left\{\frac{I_{new}(x,y)}{I_{old}(x,y)}, \frac{I_{old}(x,y)}{I_{new}(x,y)}\right\} \tag{4.2}$$

Much like simple differencing, this method was found to be overly simplistic and inadequate for high resolution data. It produced very similar results to differencing, but did sometimes produce better results on further downsized data. Since the ratio between two small numbers can be very large even their difference is very small, this algorithm produced a deceptively high response in the sea to the north, where a patch of water changed from dark to darker. The output of the image ratioing algorithm is shown in Figure 4.3.

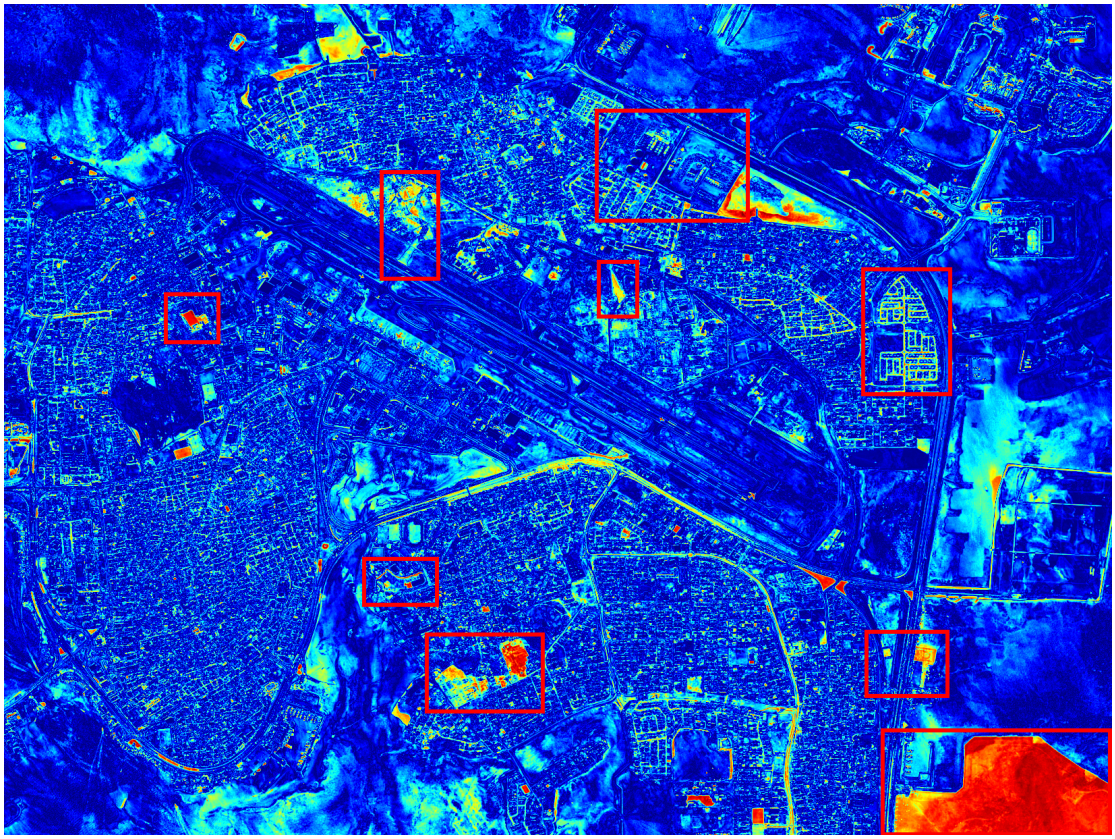


FIGURE 4.3: Change detection heat-map for image ratioing.

### 4.3.3 Visual saliency

Visual saliency is a measure of the degree to which a group of pixels stands out against its local background [32]. It is reasonable to expect an interesting change in an image to be accompanied by a change in saliency at the location of the change. For example: an empty field will have low saliency within it, but if a building is constructed in the middle of it in the more recent image then that area will now have high saliency.

We have tested this approach using a multi-scale method derived from [33]. We compute a saliency map $S(x, y)$ for the old and new images, and subtract them to compute the difference map:

$$D(x, y) = |S_{new}(x, y) - S_{old}(x, y)| \tag{4.3}$$

Saliency approaches are immune to false positives caused by uniform changes in intensity with no changes in structure or contrast. For example, if a field's intensity changes due to crop harvesting then its saliency will not change - by comparison, simple differencing or ratioing would detect this as a large and significant change. Despite this advantage, the method still produced a cluttered and uneven output which fails to draw the eye to the most important changes. In particular our implementation is very vulnerable to changes in local contrast, since it works by comparing image pixels with their local background. The output of the visual saliency algorithm is shown in Figure 4.4.

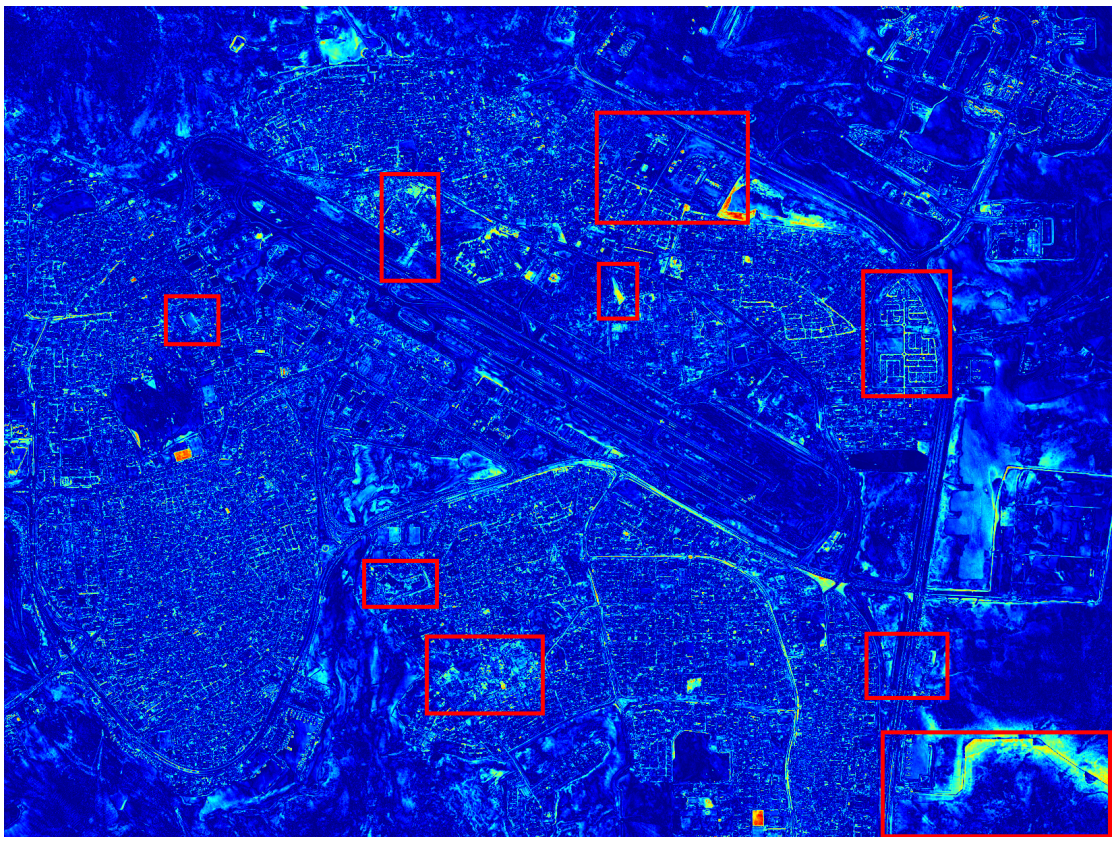


FIGURE 4.4: Change detection heat-map for visual saliency.

### 4.3.4 Gabor filters

Gabor filters are convolution masks formed by multiplying a Gaussian function with a plane wave. In two dimensions, the plane wave can have both frequency and direction, so a predefined set of filters is used - 16 in our case: four different directions in 45° increments for each of four different wavelengths. Convolving an image with a 2D Gabor filter produces a higher response in regions where that filter's frequency and orientation are more pronounced. For instance, a taxiway will produce a high response from a filter whose wavelength is similar to its width and whose orientation is perpendicular to the taxiway, while an empty field will produce a low response from all filters since it contains no intensity variation. By comparing the responses from different Gabor filters in different places rather than pixel intensities, we can measure change in a way that will not detect changes in intensity, but will detect changes in the frequency and direction of variations in intensity. We do this by convolving with a bank of 16 Gabor filters (four frequencies and four orientations), producing a 16-dimensional vector for each pixel per image. The Euclidean distance between these vectors for corresponding pixels is our change map. This is the first technique we found to respond well to the new taxiway in the north-west of the image. The output of the Gabor filters algorithm is shown in Figure 4.5.

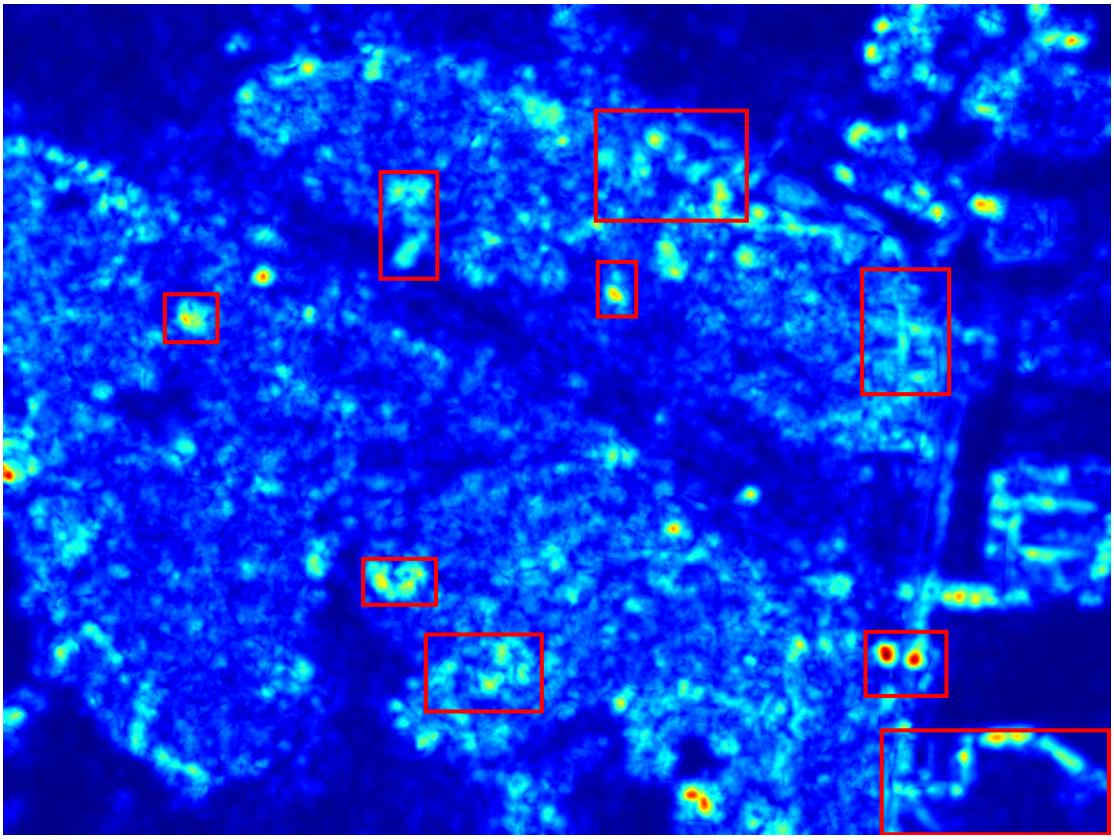


FIGURE 4.5: Change detection heat-map for Gabor filters.

### 4.3.5 Likelihood Ratio Test

These are block-based methods which first divide the images into equally sized blocks which are treated independently. For each pair of spatially corresponding blocks, two hypotheses are considered: $H_0$, that any differences between the blocks are caused by noise, and $H_1$, that a real change has occurred. The likelihood ratio is computed by modelling each block's intensities as a constant, linear or quadratic function of pixel coordinates, and measuring the deviation from these models. The parameters of these models are estimated by least-squared error fitting, and the standard deviations of the pixel intensities in blocks 1 and 2 from these models are $\sigma_1$ and $\sigma_2$, respectively. In all cases, the likelihood ratio is expressed as

$$D(x,y) = \frac{likelihood(H_1)}{likelihood(H_0)} = \frac{\sigma_0^{2n}}{\sigma_1^n \sigma^n} \tag{4.4}$$

where $\sigma_0$ is the standard deviation of both blocks from a model which attempts to describe them both, and $n$ is the number of pixels in each block. This means that each block pair has three intensity models constructed for it, each with their own parameters to be computed - one for each block and one for the joint distribution. The method is further explained in [34] and in [25]. In this subsection we show results for all three intensity models. The implementations of this method were supplied by Richard Radke [35] of Stanford University, and written by Srinivas Andra.

### Constant model

Modelling local intensity functions as constant. In this case, $\sigma_1$ and $\sigma_2$ are just the standard deviations of the intensities in the two blocks. The output of the constant model likelihood ratio test is shown in Figure 4.6.

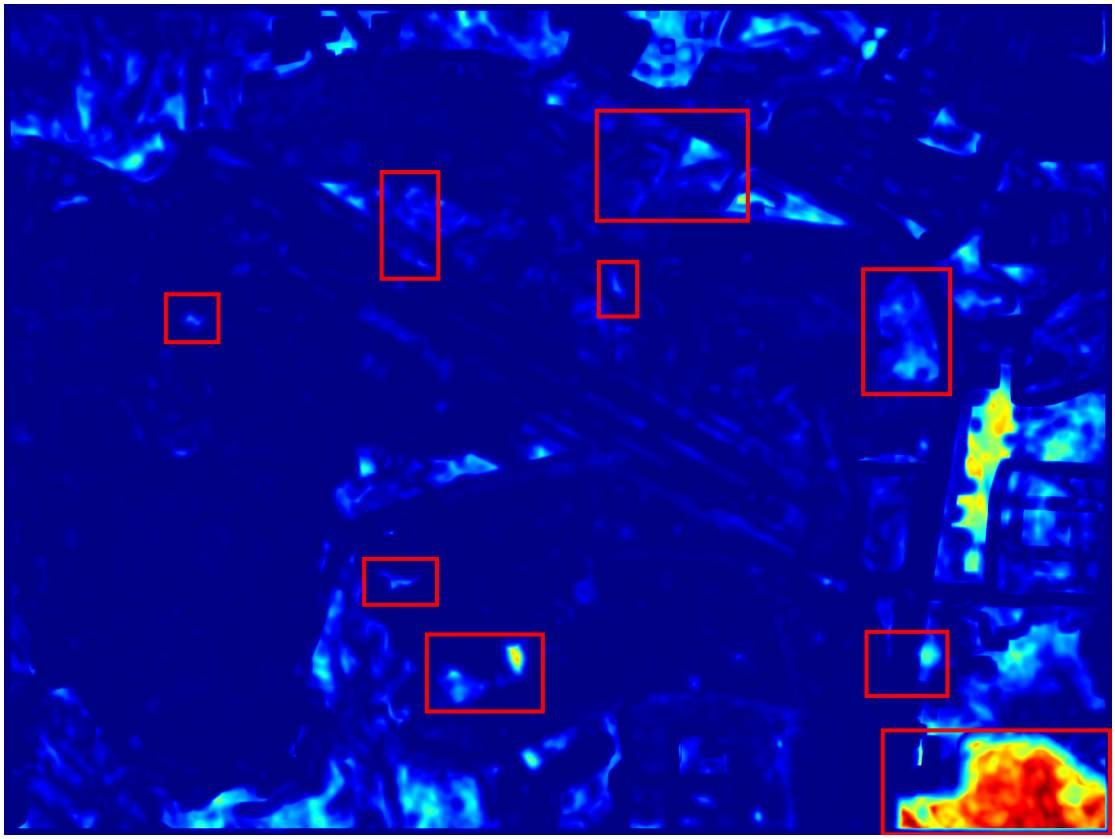


FIGURE 4.6: Change detection heat-map for likelihood ratio test, constant model.

### Linear model

Modelling local intensity functions as linear functions of $(x, y)$. Here, the three variance measures $\sigma_1, \sigma_2, \sigma_0$ are given by

$$\sigma_i^2 = \frac{1}{n} \sum_{(x,y)\in A} \left[\beta_{i1} + \beta_{i2}x + \beta_{i3}y - g(x, y)\right]^2 \tag{4.5}$$

where the coefficients $\beta_{1i}$, $\beta_{2i}$, $\beta_{3i}$ describe the linear model of the intensity function of the spatial region $A$ covered by the blocks, obtained by a least squared error fit. The output of the linear model likelihood ratio test is shown in Figure 4.7.

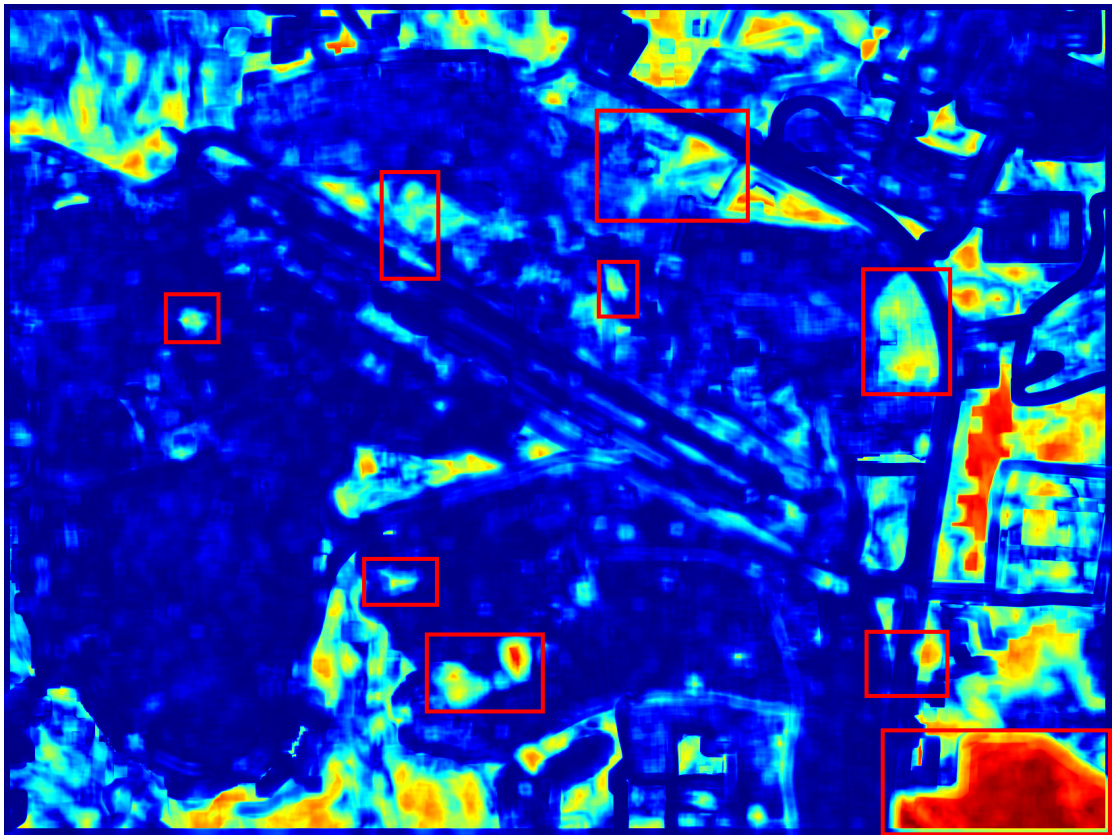


FIGURE 4.7: Change detection heat-map for likelihood ratio test, linear model.

### Quadratic model

Modelling local intensity functions as quadratic functions of $(x, y)$. Here, the three variance measures $\sigma_1, \sigma_2, \sigma_0$ are given by

$$\sigma_i^2 = \frac{1}{n} \sum_{(x,y)\in A} \left[\beta_{1i} + \beta_{2i}x + \beta_{3i}y + \beta_{4i}x^2 + \beta_{5i}y^2 + \beta_{6i}xy - g(x, y)\right]^2 \tag{4.6}$$

where the coefficients $\beta_1$, $\beta_2$, $\beta_3$, $\beta_4$, $\beta_5$, $\beta_6$ describe the quadratic model of the intensity function of the spatial region $A$ covered by the blocks, obtained by a least squared error fit. The output of the quadratic model likelihood ratio test is shown in Figure 4.8.

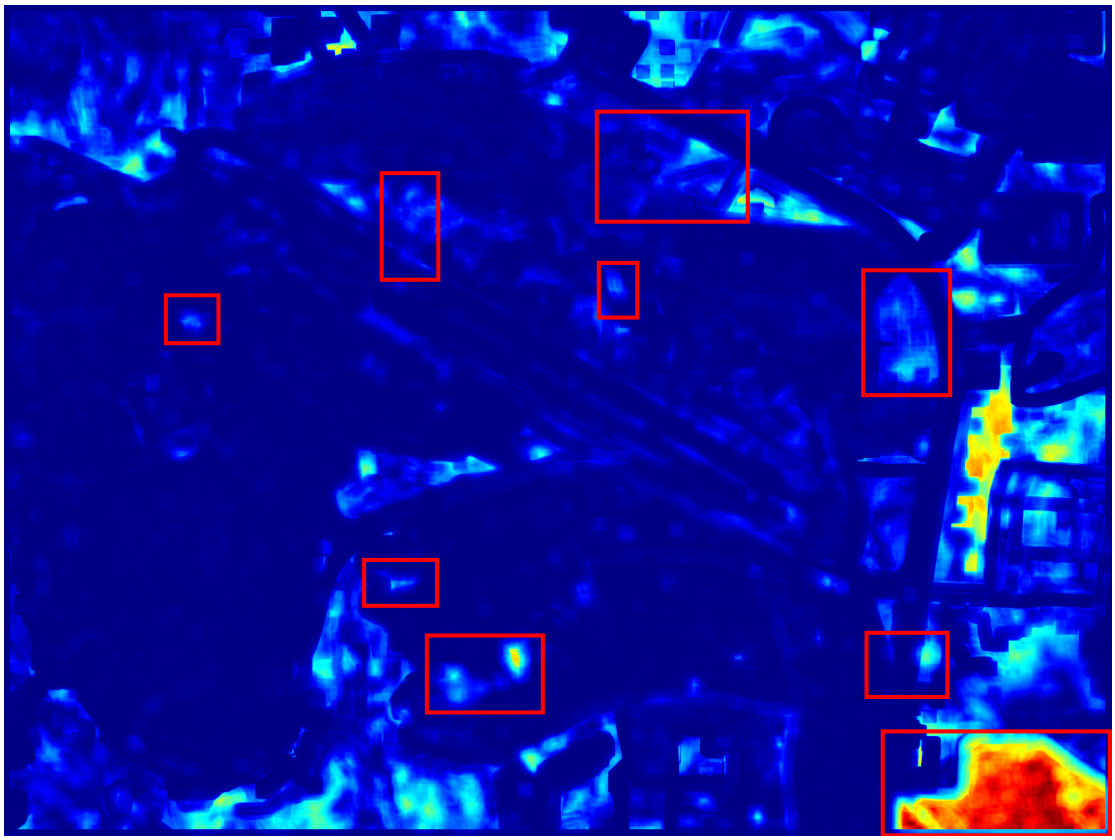


FIGURE 4.8: Change detection heat-map for likelihood ratio test, quadratic model.

### 4.3.6 Derivative comparison

This method was proposed by Skifstad and Jain [36] as a logical extension to the quadratic model developed by Hsu [34]. By considering the differences between partial derivatives of the quadratic model, it achieves more robustness against changes in illumination. The output of the derivative comparison algorithm is shown in Figure 4.9. The implementation of this method was supplied by Richard Radke [35] of Stanford University, and written by Srinivas Andra.

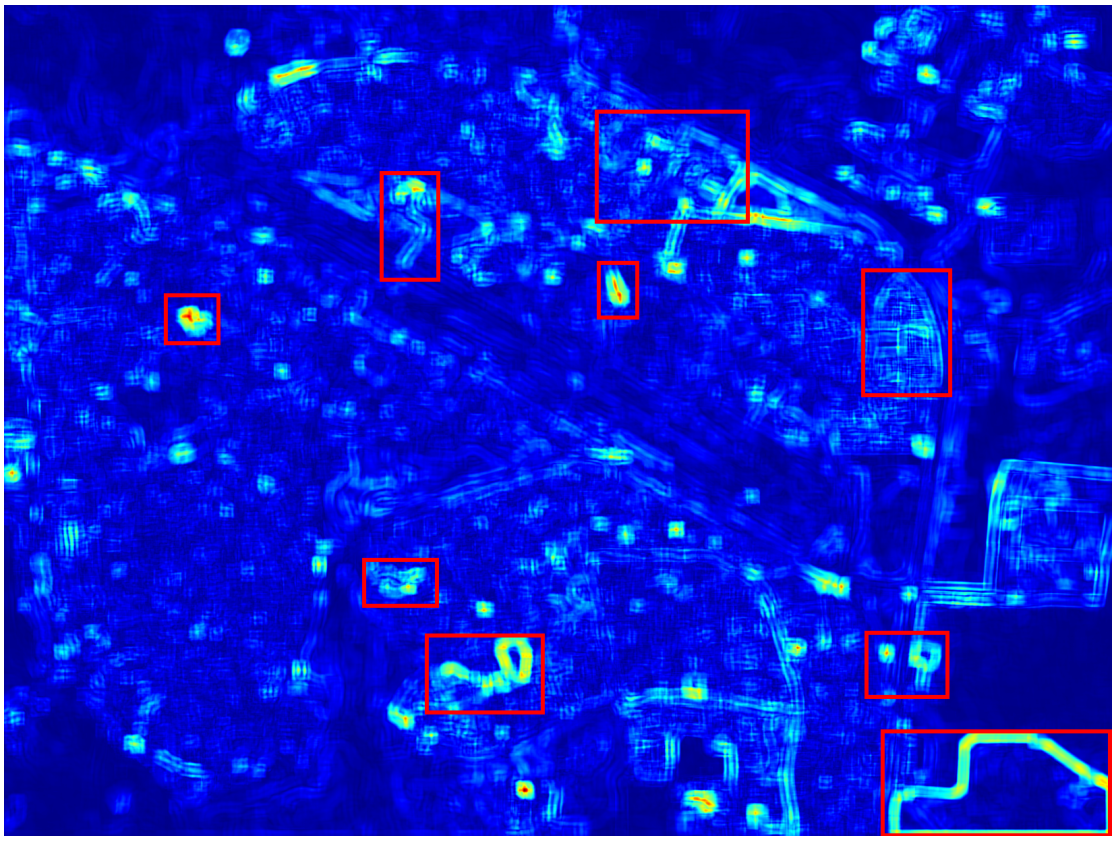


FIGURE 4.9: Change detection heat-map for derivative comparison.

### 4.3.7 Linear dependence test

The other method proposed in [36]. It calculates the difference for each image block as the variance of the ratio of corresponding pixel intensities within that block. Pixel intensity can be modelled as the product of local illumination brightness and the reflectance of the surface at that point, and illumination can be assumed constant within a block. Therefore this method will not respond when the only change is a change in illumination (since all pixels in the block will brighten or darken by the same factor). The output of the linear dependence test is algorithm is shown in Figure 4.10. The implementation of this method was supplied by Richard Radke [35] of Stanford University, and written by Srinivas Andra.

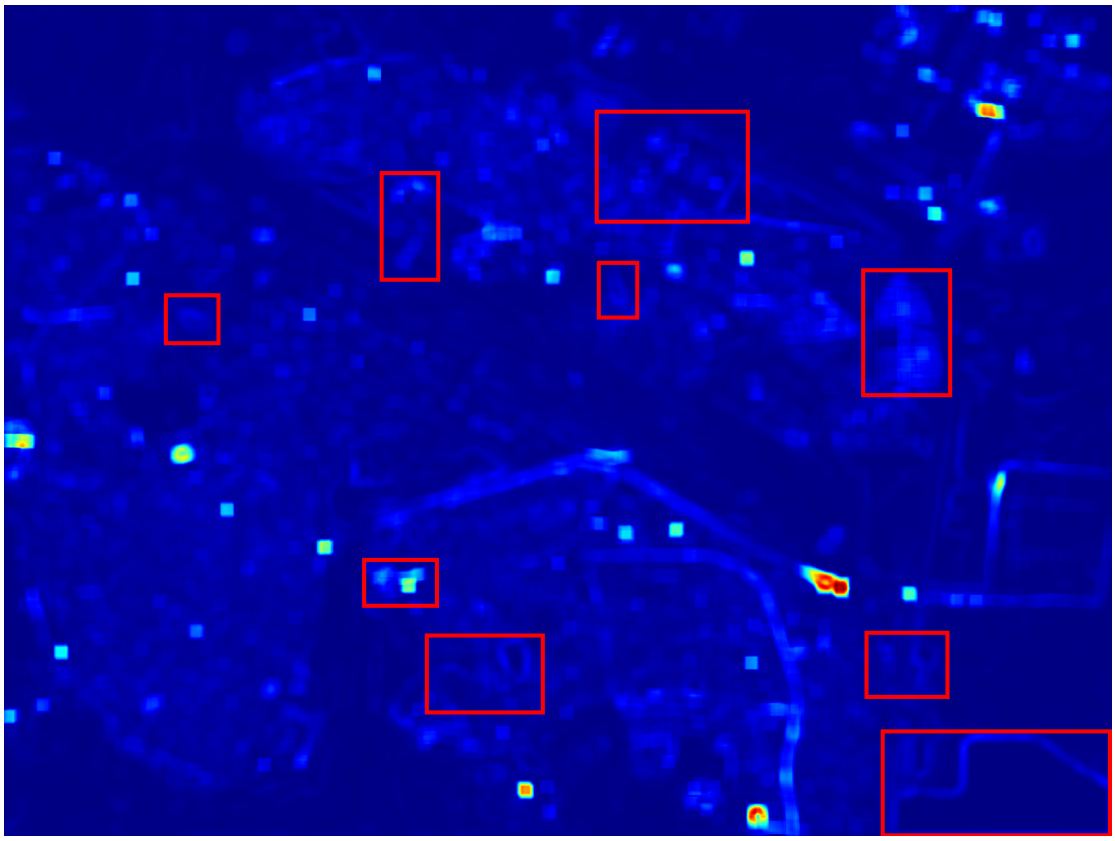


FIGURE 4.10: Change detection heat-map for linear dependence test.

### 4.3.8 Wronskian method

The mathematics of the Wronskian method and its application to change detection are derived in [37]. In practice, it is implemented as a block based algorithm, which for each block calculates the following value:

$$D = \frac{1}{n}\sum_{i=1}^{n}\frac{x_i^2}{y_i^2} - \frac{1}{n}\sum_{i=1}^{n}\frac{x_i}{y_i} \tag{4.7}$$

where $x_i$ and $y_i$ are the intensities of the $i$th pixels of the block in the new and old image, respectively. The output of the Wronskian method algorithm is shown in Figure 4.11. The implementation of this method was supplied by Richard Radke [35] of Stanford University, and written by Srinivas Andra.

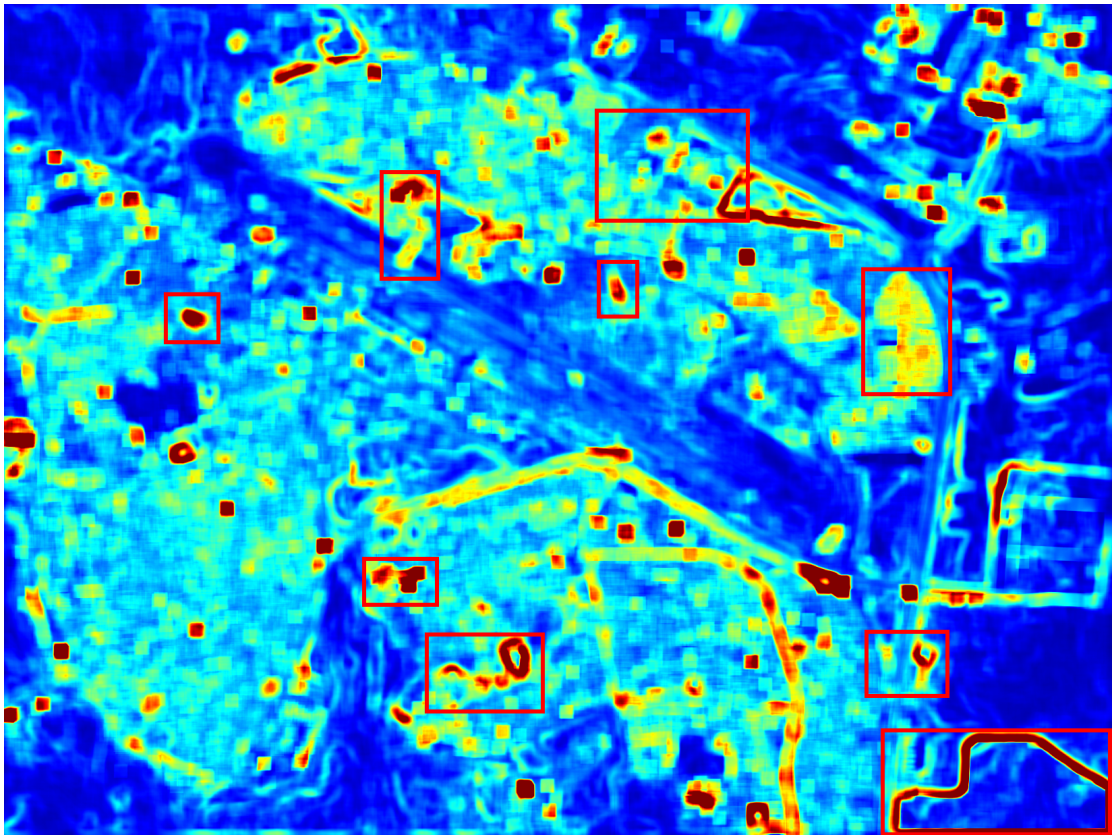


FIGURE 4.11: Change detection heat-map for Wronskian method.

### 4.3.9 Quotient Variance Re-implementation

The same concept as in the linear dependence test, but re-implemented using image convolutions rather than a block-based approach. The previous implementation computed the variance of the intensity quotient for every (overlapping) square region within the image. The new implementation instead computes the variance in a gaussian window by convolution with gaussian kernels. This approach runs much faster, and produces better results with no blocking artifacts. The output of the quotient variance re-implementation is shown in Figure 4.12.

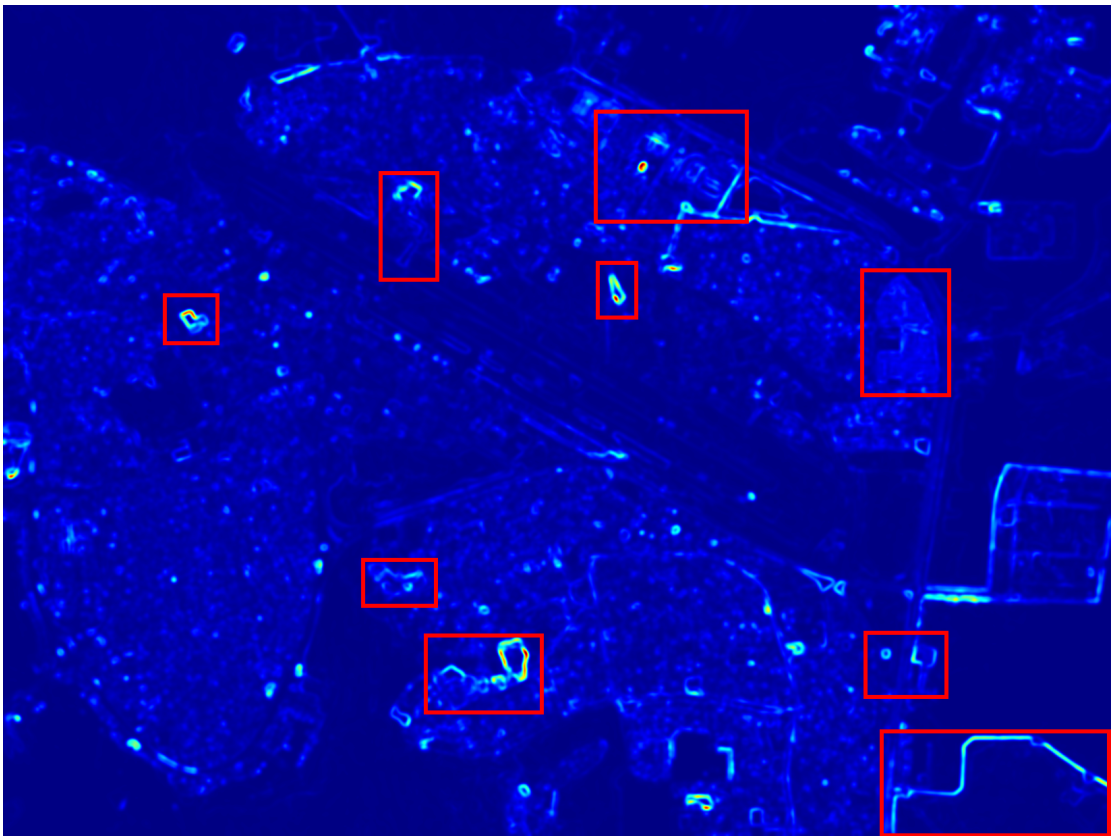


FIGURE 4.12: Change detection heat-map for quotient variance re-implementation.

### 4.3.10 Gabor phase difference

Inspired by the use of Gabor filter response phases in iris recognition, we tested a modified Gabor change detector which looked for changes in response phase rather than magnitude. Although this approach consistently produced a high response in important change regions, it also gave numerous false positives. The output of the Gabor phase difference algorithm is shown in Figure 4.13.

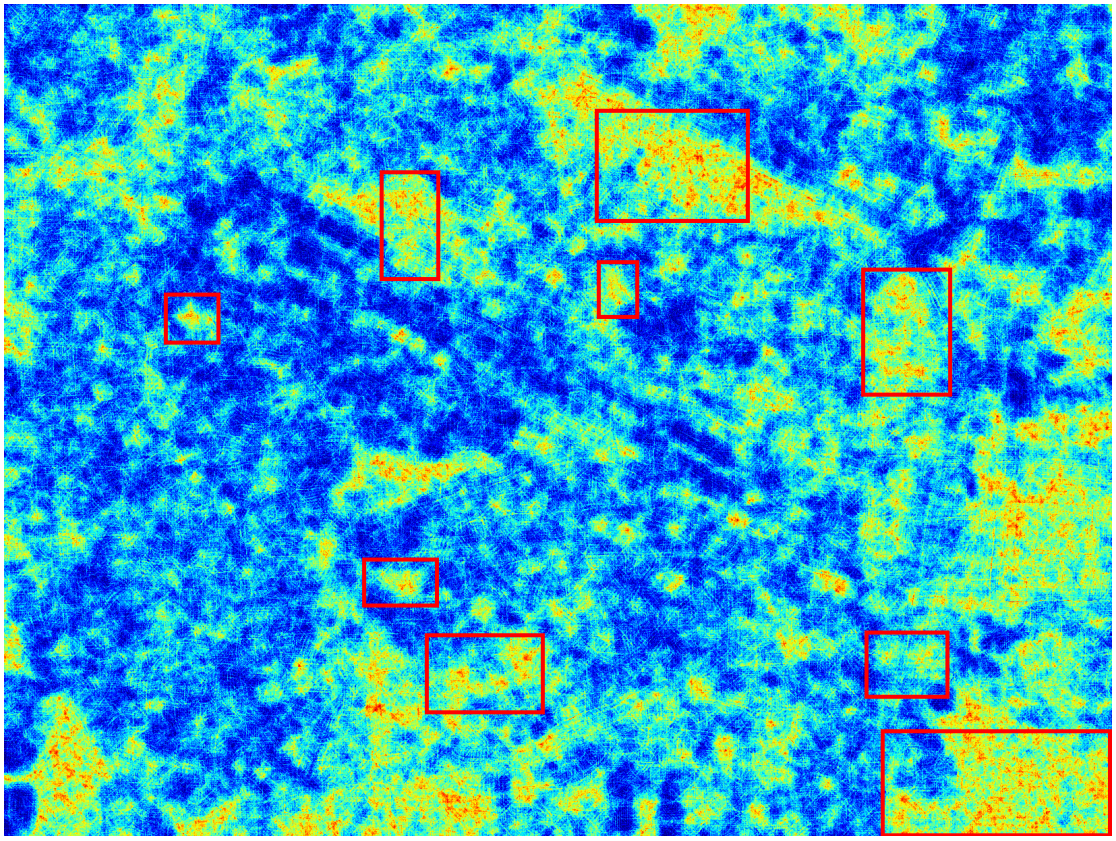


FIGURE 4.13: Change detection heat-map for Gabor phase difference.

### 4.3.11 Spatial frequency difference

Since changes in structure are associated more with a change in intensity frequency rather than direction, we tried replacing the Gabor wavelet bank of four frequencies in four directions each, with 16 undirected wavelets of different frequencies. This trades direction information for more accurate measurement of local frequencies. Currently the difference measure is computed by summing the absolute difference in response strength across all 16 frequencies, but there are many other ways in which the responses could be compared, some of which may produce better results. The output of the spatial frequency difference algorithm is shown in Figure 4.14.

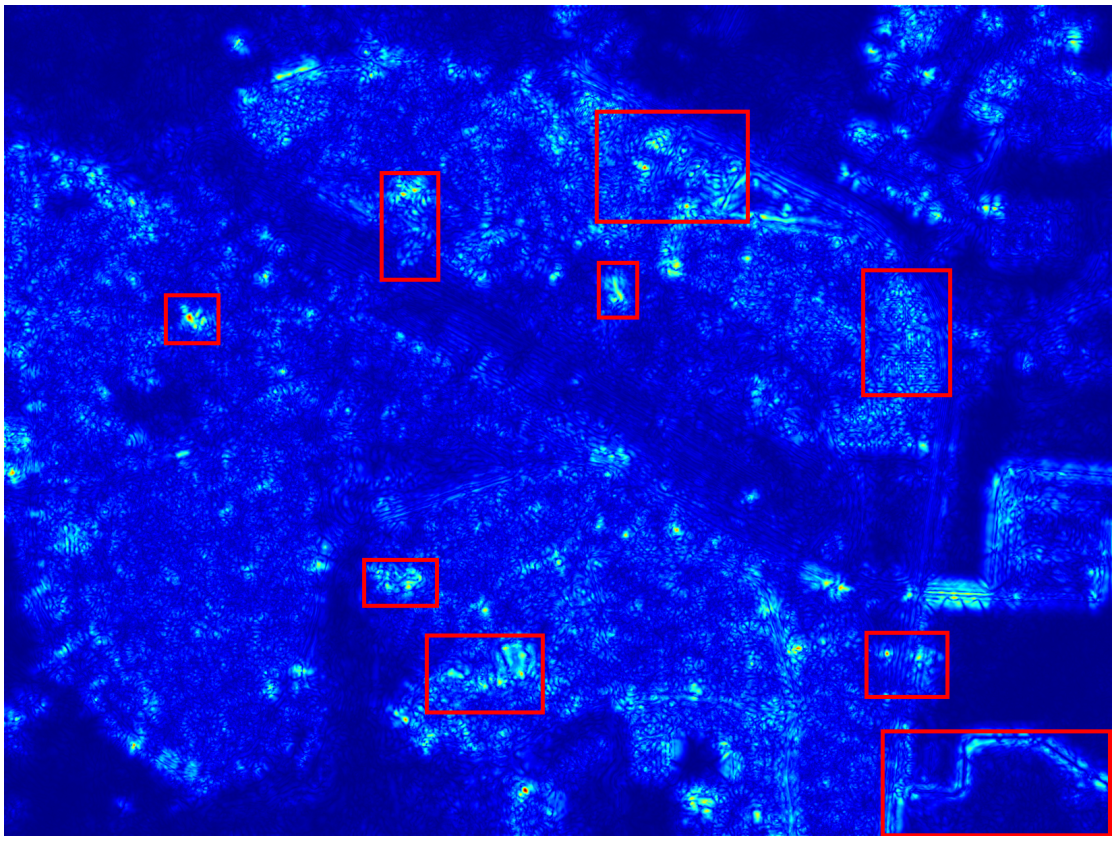


FIGURE 4.14: Change detection heat-map for spatial frequency difference.

### 4.3.12 Sum of first derivative

Important features often appear as dark shapes on light backgrounds, or vice-versa. To measure the presence of such features, we look for areas where intensity changes a lot, by computing a local sum of intensity gradient magnitude. However, if a feature is light on dark or vice-versa, then the local sum of the signed intensity gradient should be close to zero, since the intensity falls within the object then rises back to background level on the other side. We compute a saliency measure by calculating the degree to which both these conditions are met, and detect changes by displaying the difference of this measure between the two images. The output of the sum of first derivative method is shown in Figure 4.15.

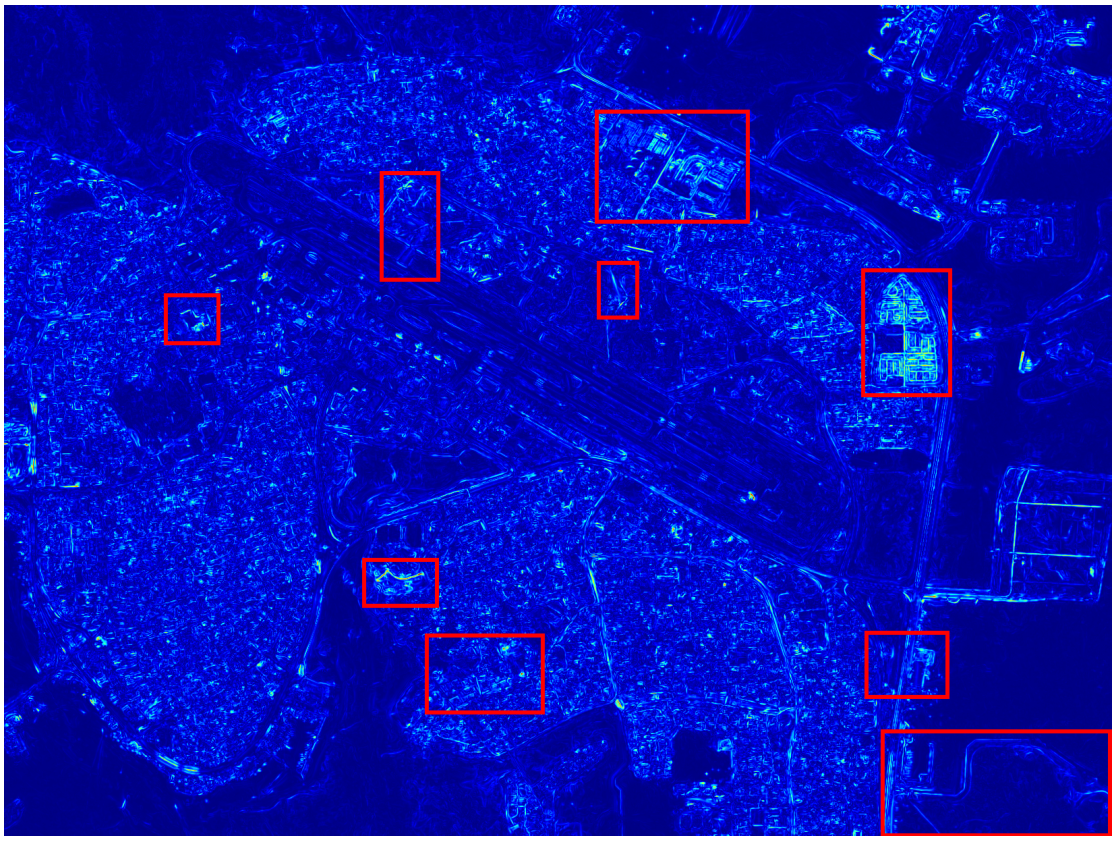


FIGURE 4.15: Change detection heat-map for sum of first derivative.

### 4.3.13 Sum of second derivative

We compute a local sum of the second derivative of intensity at every point in the image, by convolving first with a discrete Laplacian kernel, then a Gaussian kernel. This operation can be thought of as measuring the roughness of different regions. By seeing how this roughness measure changes from one image to the next, we produce a change measurement that responds strongly to high frequency changes, such as the addition or removal of small buildings. This approach may be able to detect low frequency changes (e.g. larger features such as taxiways), if it is applied at multiple scale levels. The output of the sum of second derivative method is shown in Figure 4.16.

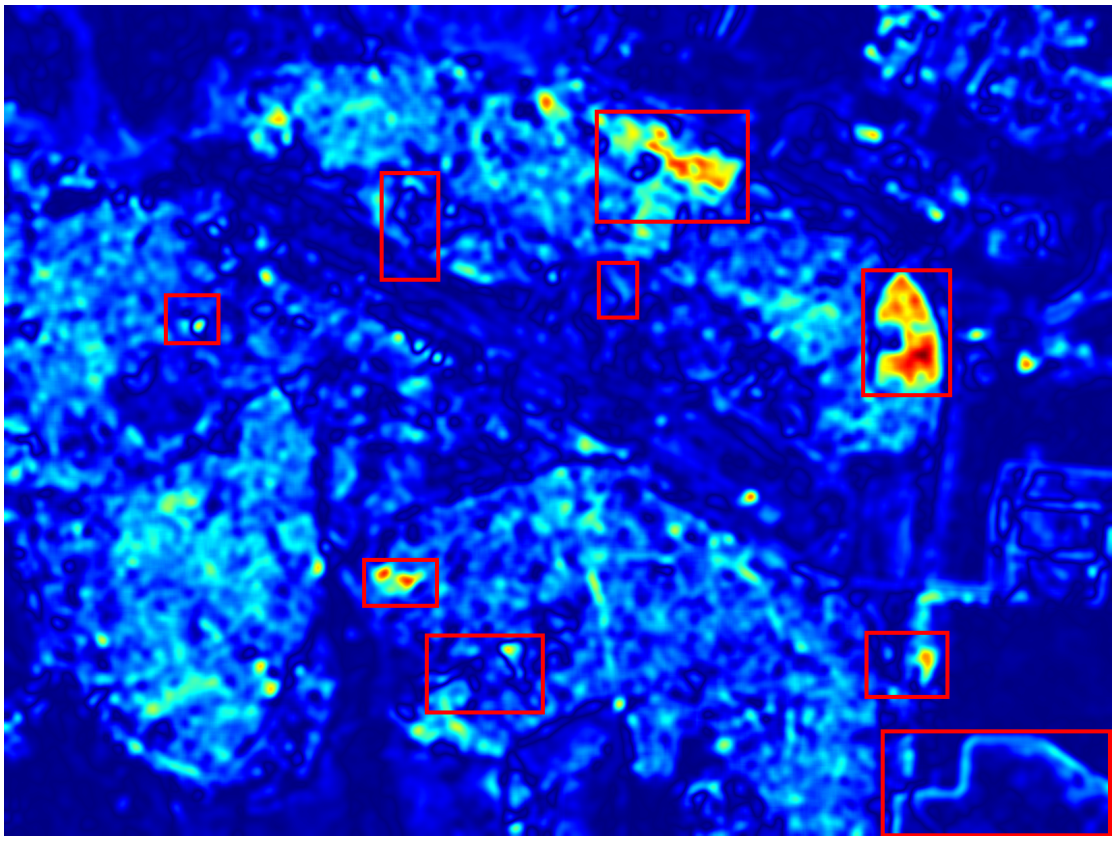


FIGURE 4.16: Change detection heat-map for sum of second derivative.

### 4.3.14 Local ternary patterns

Most of the methods previously demonstrated have been susceptible to false positives caused by changes in local contrast. For instance, if in the new image the aiming point markings on a runway stand out brighter due to repainting or changes in illumination, then many algorithms would detect this as a change, even though no change in shape has occurred. Local area change avoids this problem by classifying the local neighbours of each pixel into three classes: significantly brighter, darker, or roughly the same intensity. This is similar to local binary patterns, which describe local texture by producing bitstrings describing which of a pixel's neighbours have greater intensity. By introducing an "equal intensity" class (which we define as intensity difference being less than the global standard deviation of intensity), we avoid the problem of large, random changes to bitstrings in homogeneous regions caused by small changes in pixel intensity. The change response for each pixel is calculated by counting the number of local neighbours of that pixel who change from one class to another. In this way, if the local contrast increases and a pixels brighter neighbours become even brighter relative to that pixel, then no change is detected, since they were already classified as brighter. Additionally, we improve detection of different sized changes by taking the arithmetic mean of the algorithm's response at different scale levels. The output of the local ternary pattern algorithm is shown in Figure 4.17. We also give a demonstration of this method's invariance to contrast changes in Figure 4.18.

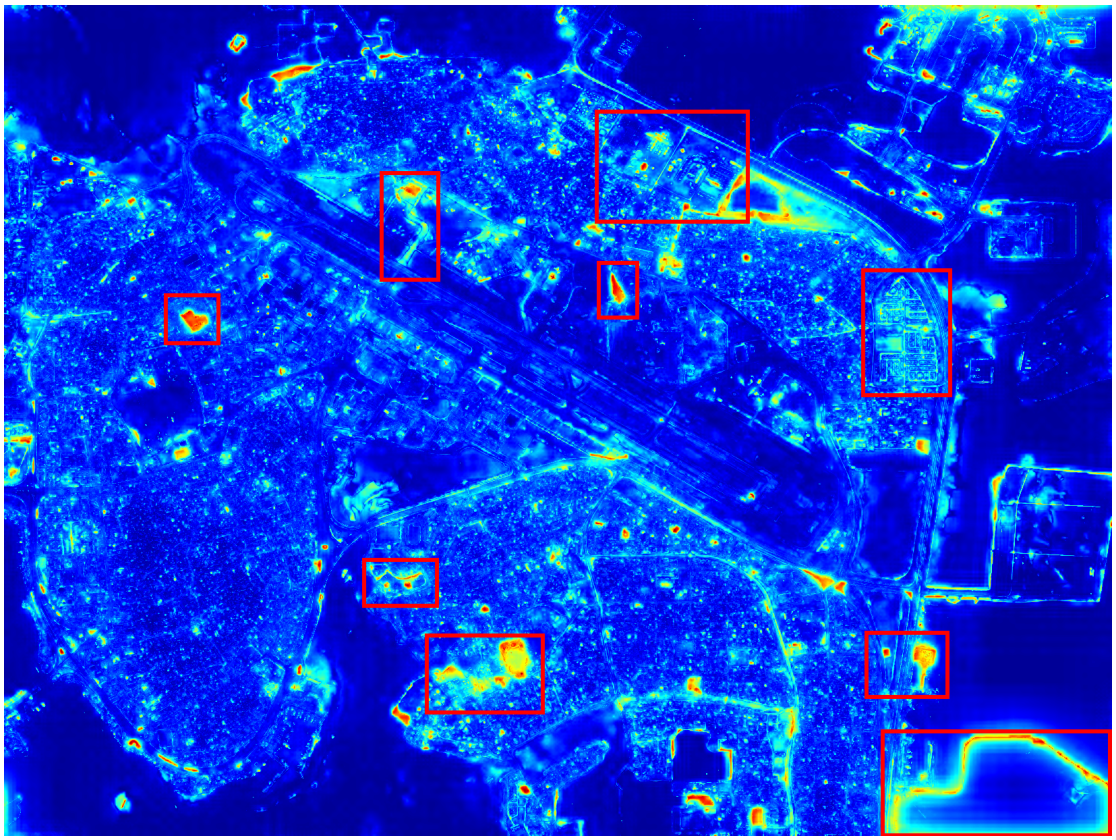


FIGURE 4.17: Change detection heat-map for local ternary patterns.

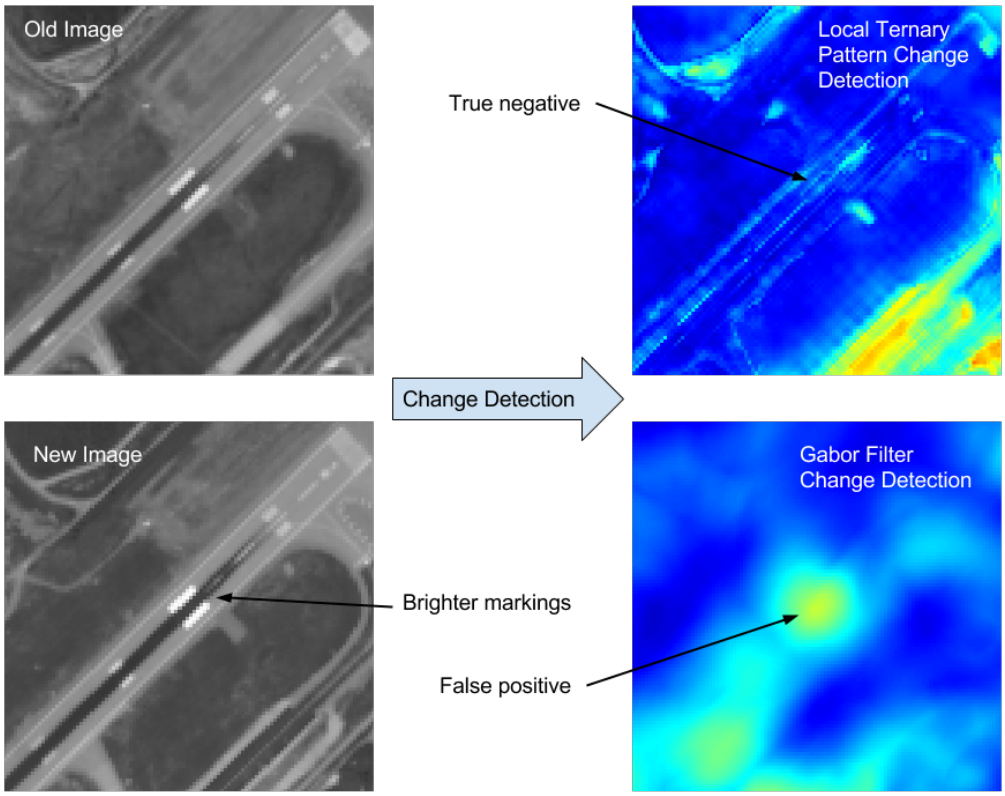



FIGURE 4.18: This figure demonstrates the contrast invariance of the Local Ternary Pattern method. In one of the runways in Helsinki airport, the aiming point markings at one end appear brighter in the new image than in the old. This does not constitute a meaningful change and we wish to ignore it, but many change detection algorithms (such as our Gabor filter method, pictured in the bottom right) produce a false positive here. But because Local Ternary Patterns classify the markings as brighter than their surroundings in both the old and new image, no false positive is produced.

## 4.4 Conclusion

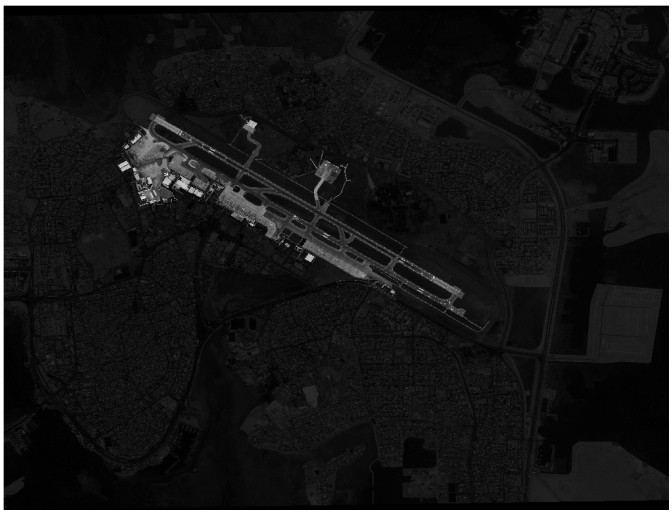
Change detection algorithms are a promising research area for this project, since they offer to produce saliency heat-maps which can direct the attention of either human operators or other algorithms to areas of significant change. Numerous difficulties were encountered in this section; many of the algorithms tested produced false positives in empty areas such as bodies of water, while detecting the added taxiway was surprisingly difficult for such an apparently obvious feature. False positives also occurred due to effects such as specular reflection on building roofs caused by changing illumination. It seems that much of the remote sensing change detection literature is focussed on lower resolution image data than ours, or on simpler problems such as measuring vegetation growth, and so performs poorly on our images. Gabor filters and the local ternary pattern method seemed to produce the most promising results, since they yielded the most noticeable response to the new taxiway and fewest false positives; this may make them useful to a human operator. Local ternary patten seemed particularly promising as they highlighted changes in sharp detail (see Figure 4.17), and does not

produce false positives due to change in brightness in the absence of change in shape (see Figure 4.18).

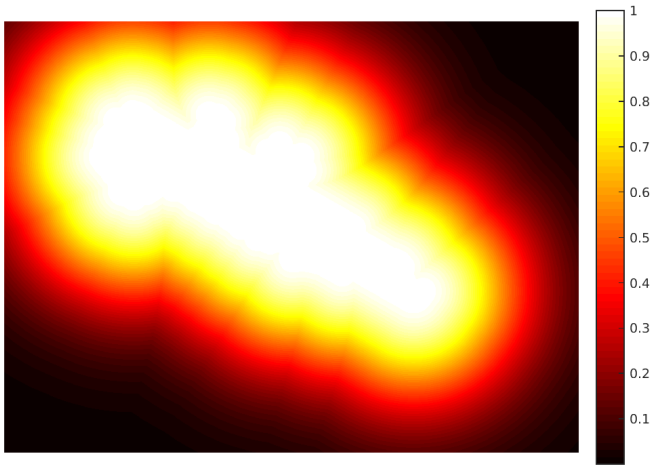
The intended use of these heat-maps is in assisting a human expert in quickly locating regions of significant change, by drawing the eye to the appropriate region without having to compare two images manually. This makes the efficacy of the output inherently subjective, and therefore difficult to validate quantitatively. It would have been possible to use AMDB shapefiles as ground truth, but these algorithms cannot distinguish between changes to airport features and irrelevant changes. Therefore this would have forced us to either unfairly penalise algorithms for correctly detecting changes that are not in the AMDB, or ignore all false positives. Instead, we have assessed their performance qualitatively, by observing their response in manually chosen regions of the test image where significant changes occur. We summarise these findings in Table 4.1.

Given how many small changes are normally present in these images, the approach of producing change detection heat-maps may always be of limited use, since so many irrelevant changes are detected. However, it is possible to suppress irrelevant changes to some extent, by considering changes further away from the airport as less important, as Figure 4.19 shows.

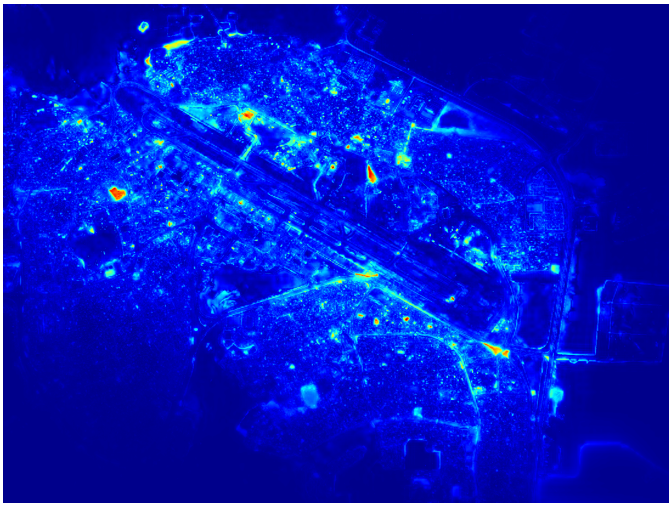
Object based change detection may be the most fruitful path for further research. Literature reviews by Blaschke *et al.* [38] and Chen *et al.* [39] provide overviews of many such algorithms. These approaches may be more general purpose than Jeppesen requires, since we are interested specifically in detecting changes in AMDB features, but they are still likely to contain many applicable ideas. More possibilities for automated change detection will emerge if a reliable process for sub-pixel accuracy image registration can be found, since this chapter was limited to coarse change detection in the absence of fully accurate registration. For example, sub-pixel registration would make it possible to detect changes in fine structures such as the taxilines shown in Figure 3.2 using the same or similar change detection techniques outlined in this chapter.



(A)




(B)



(C)

FIGURE 4.19: (a) Bahrain airport segmented from its surroundings by overlaying GIS polygons on the image. (b) A distance weighting function on that image, with a Gaussian falloff (standard deviation equal to the minor axis length of an ellipse with the same second moments as the airport mask). (c) The local ternary pattern change heat-map (Figure 4.17) multiplied element-wise with (b), to suppress changes which occur far from the airport.

TABLE 4.1: Qualitative summaries of the change detection techniques assessed in this chapter.

| **Method** | **Description** |
|---|---|
| Simple differencing | Sufficient to detect many changes, but response is proportional to change in brightness rather than shape. Responded weakly to new taxiway. |
| Image ratioing | Unsuitable; gave a highly cluttered response with many false positives in dark regions, e.g. water, where intensity ratios become high despite small absolute differences. |
| Visual saliency | Unsuitable; gave too uniform a response due to the densely detailed nature of the images. Is better suited to detecting isolated objects against uniform backgrounds. |
| Gabor filters | Detected most of the target changes, but had a high background response too. Responded noticeably to new taxiway. |
| Likelihood ratio test | Unsuitable; all three variants yielded unacceptable false positives, particularly in water. |
| Derivative comparison | Detected most of the target changes, but had a high background response too. Responded noticeably to new taxiway. |
| Linear dependence test | Unsuitable; yielded low response on almost all parts of the image, failed to respond to most changes |
| Wronskian method | Unsuitable; responses to target changes were difficult to discern amongst unacceptably high background response. |
| Quotient Variance Re-implementation | Small but noticeable response in most target regions, low background response. Responded weakly to new taxiway. |
| Gabor phase difference | Unsuitable; unacceptably high background response. |
| Spatial frequency difference | Produced noticeable response to most target regions, but responded weakly to new taxiway. Similar technique to Gabor filters but showed lower background response. |
| Sum of first derivative | Unsuitable; failed to detect most target changes. |
| Sum of second derivative | Unsuitable; detected some changes much better than most other methods but failed to detect new taxiway and yielded high background response. |
| Local ternary patterns | Produced the clearest response to the new taxiway. Highlights changes in sharp detail and is more sensitive to change in shape than to change in contrast or brightness. |

# Chapter 5

# Line Enhancement

## 5.1 Introduction

Several of the features in Jeppesen's AMDB systems take the form of thin lines painted on a darker surface, mostly with the purpose of guiding taxiing planes around the airport (see Figure 5.1). These make for promising candidates for automatic detection, since there has already been a great deal of research focussed on the more general problem of extracting curvilinear features (such as blood vessels or nerve fibres) from bio-medical data. In principal, bright lines on dark backgrounds can be described by a simple mathematical pattern of pixel intensities, unlike most other AMDB features. In addition, AMDB lines mostly have a constant thickness, unlike objects such as blood vessels, which have variable thickness. This means that we can process our images at a single scale level, in contrast with most biomedical literature where a multi-scale approach is necessary. Compared to larger, more complex AMDB features such as aprons, taxiways and buildings, these bright painted lines on smooth dark backgrounds are quite attainable from an image processing perspective.

They may also be useful as a precursor to detecting other features, since the taxiline network forms a kind of skeleton which connects all the main airport features together. Every taxiway should have a taxiline down its centre, so if those centrelines are detected first then extracting the boundaries of the taxiways should become much easier. Furthermore, the taxilines should eventually lead to the apron if followed - at which point they tend to branch, and terminate as standing lines (see Figure 5.1c). The goal of this chapter is to develop a system which can take a bi-temporal pair of airport images, and ultimately output a binary image showing where lines have appeared in the new image which were not present in the old one.

The AMDB features which correspond to visible linear structures in the images are:

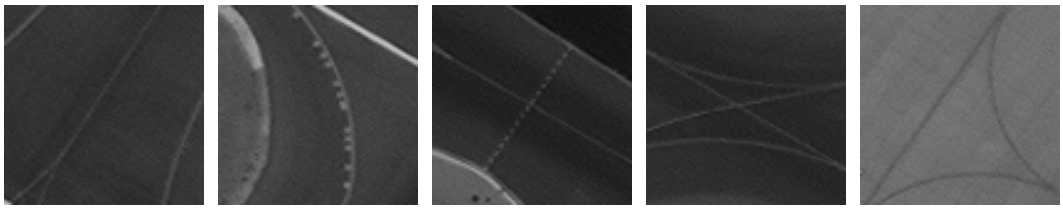
- txiline.shp - centreline markings to guide aircraft along taxiways
- txiint.shp - taxiway intersection marking
- stopbar.shp - hold position marking, in the form of a perpendicular line across a taxiway
- stndline.shp - these guide aircraft to their parking positions on the apron
- lahso.shp - runway marking indicating Land And Hold Short Operations
- exit_ln.shp - a lines guiding aircraft from taxiways onto runways

Figure 5.2 shows the variety of styles in which these AMDB lines manifest themselves. An ideal line detector would be able to detect all of these styles of line equally well; in practice we find achieving such a level of generality to be one of the main challenges of this work.

We do not attempt to distinguish between lines which correspond to different AMDB features, since they mostly differ in functional purpose rather than visual appearance.

(A) txiline.shp (B) stopbar.shp (C) stndline.shp

FIGURE 5.1: Examples of a taxiline, stopbar and standing line as represented by Bahrain airport's AMDB shapefiles. The shapefiles have been overlaid on the image as red lines.



(A) (B) (C) (D) (E)

FIGURE 5.2: Airport lines can take on a variety of appearances.

To distinguish them would require analysis of the line's high-level context, which is a very different problem and beyond the immediate scope of this chapter. The research objective at this stage was to produce binary line maps (binary images of equal size to the input image, where white pixels indicate the presence of a line), which could later be used for post classification change detection. Our approach to this problem mostly consists of implementing a variety of metrics which enhance linear structures while suppressing others, producing greyscale response image from which a binary line map can be extracted by thresholding. In this chapter, we describe the approaches we implemented, the challenges we met and the results we have attained so far.

## 5.2 Related Work

A large body of work exists concerning curvilinear structure detection, much of it in the medical domain focussing on blood vessel analysis [40], because of its diagnostic value for many diseases. Other applications include characterisation of neural tissue [41] and detection of roads in aerial or remote sensing imagery [42]. One of the most widely cited contributions in this field was the "vesselness" curvilinear enhancement filter by Frangi *et al.* [43]. This method computes Hessian matrix eigenvalues at every pixel in an image and expresses the "vesselness" of each pixel as a function of those eigenvalues. This vesselness metric produces a high response at the centre of tubular structures, which can be thresholded to produce a binary vessel map. The original paper proposes a multi-scale approach which can detect vessels of different thicknesses,

and the method is defined for both 2D and 3D data. This approach has been quite influential and is expanded on several times by other authors [44] [45] [46]. A more detailed explanation of the central idea, and its application to our work, are discussed in the method section.

Much of the work surrounding vessel extraction is focussed on 3D data, from imaging devices such as fMRI and confocal microscopy. With regard to 2D data, there has been much attention on automatically segmenting retinal blood vessels, with the aim of developing fast, inexpensive screening processes for many diseases, for mass deployment without the need for thousands of trained ophthalmologists. Mendonça *et al.* [47] developed a method for segmenting these vessels by first detecting their centrelines, then broadening those lines to the thickness of the vessels using morphological reconstruction. This paper described a multi-step technique employing a variety of techniques including background normalisation and a modified top hat transform. Chaudhuri *et al.* [48] proposed the important technique matched filter response, in which the image is convolved with a set of oriented kernels, each of which resemble piecewise linear segments of a blood vessel. This is essentially the same idea as template matching in the field of object recognition. These filters enhance the blood vessels by responding strongly where their scale and orientation matches that of a blood vessel. Other approaches to retinal vessel extraction include application of support vector machines for supervised learning classification [49], and an extension of matched filter response which uses region based network properties in conjunction with local properties of the matched filter response [50].

## 5.3 Line Enhancement Metrics

In theory, any pixel forming part of a line should be brighter than its neighbours either side in two anti-parallel directions, and the same intensity as its neighbours in the orthogonal directions. We began by producing line enhancement functions which attempted to recognise this "ideal line" pattern. In most cases, the idea is to compute some "lineness" metric for every pixel in a full resolution airport image, which should be greater for pixels which are more likely to belong to a line. This metric can then be thresholded to produce a binary image showing where the lines are. With two such binary line maps produced from co-registered airport images, a binary XOR should produce a binary change map showing where lines have appeared or disappeared. This would be a form of per-pixel change detection. Alternatively, one could take an object-based change detection approach: extract line objects by finding connected components in the binary images, then attempt to match these objects based on morphological properties such as tortuosity, position and the number and length of branches.

We have implemented a set of seven line enhancement methods. Some of these features were derived from standard image processing techniques or papers in bioimage processing, others were original. To compare them, we will show as heat-maps their responses to a sample region of Bahrain International Airport (see Figure 5.3). This region shows many kinds of feature, some of which we wish to detect, others to ignore.

### 5.3.1 Gradient Vector Alignment

This was a novel technique, motivated by considering the distribution of intensity gradients around an ideal line. In an ideal line, the intensity gradient vectors all point in

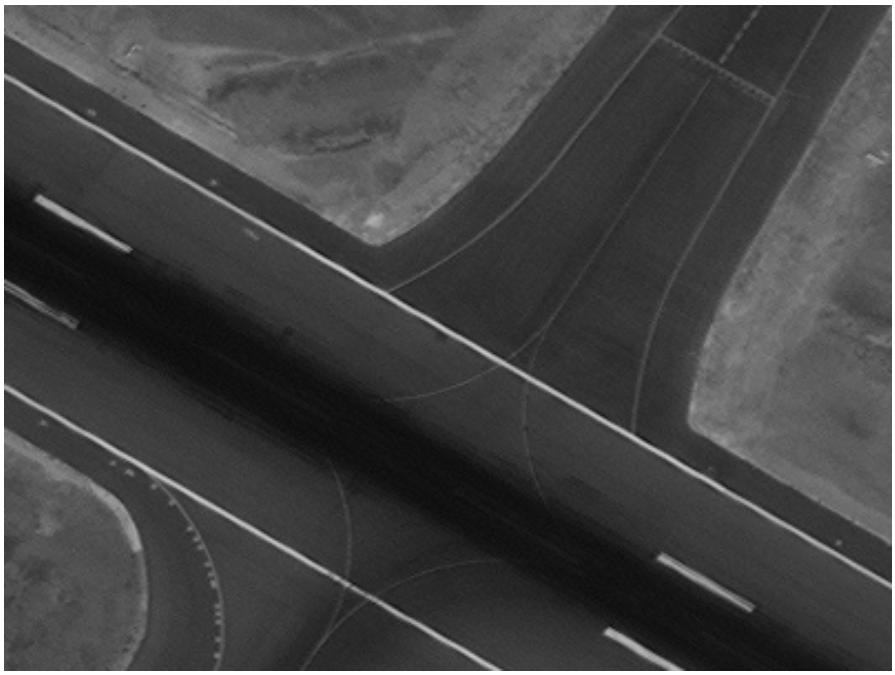


FIGURE 5.3: A sample region of Bahrain International Airport. The runway's side-stripes are bright, high-contrast lines, while the taxiway's center-line is much fainter. A dashed line is visible near the top-right, while a line with small blobs on it is present in the bottom-left. Junctions are visible in the top-right and where the taxiway's lines join the runway side-stripes. The curved exit lines are also visible, one of which is slightly broken.

two anti-parallel directions, inwards towards the centre of the line (or outwards if the line is dark on bright). An edge, by contrast, will have gradient vectors pointing in only one direction. Our images contain many edges, such as the boundary between the taxiway shoulder and surrounding terrain in Figure 5.3, so we require a technique which can ignore these features. To detect this "ideal line" gradient pattern, we first represent the intensity gradient vectors as a complex numbers:

$$Z = \frac{\partial I}{\partial x} + \frac{\partial I}{\partial y} i \tag{5.1}$$

where $I(x, y)$ is the image intensity and $i$ is the imaginary unit. Complex numbers have the useful property that $(-Z)^2 = Z^2$. This means that if we sum the gradient vectors in a small region centred on a line pixel, the values of $Z$ will cancel out since they point in opposite directions, while the values of $Z^2$ will reinforce since they point in the same direction. We can perform a local summation at every point in an image using a convolution. The full formula for this metric then is

$$L = |Z^2 * g| - |Z * g| \tag{5.2}$$

where $L$ is our metric response, $*$ denotes convolution and $g$ is a Gaussian kernel with standard deviation equal to one pixel - roughly half the width of the lines we are searching for. The output of this metric on our sample region is shown in Figure 5.4.

As with most metrics its response is proportional to the contrast of the line, so the runway side-stripes appear much more strongly. The response to the dashed line was

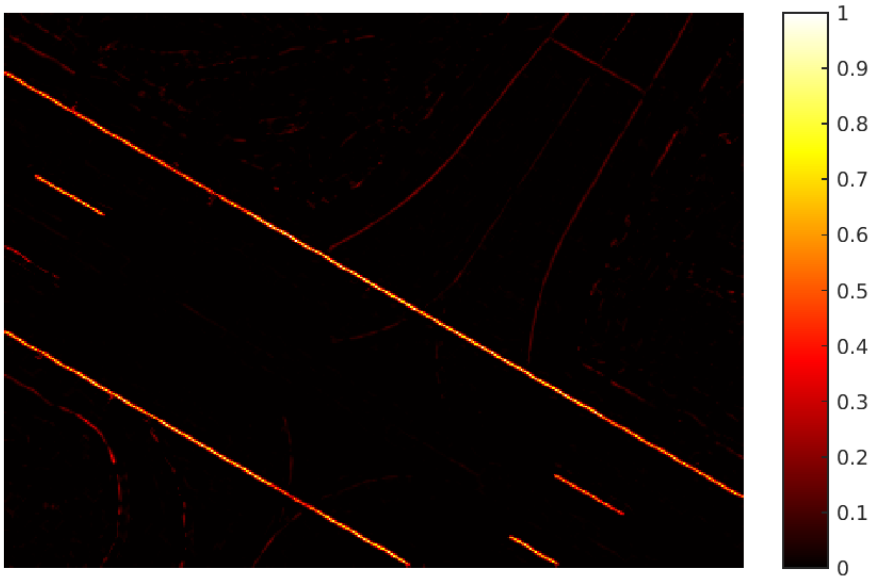



FIGURE 5.4: Output of gradient vector alignment metric. The colorbar shown here applies to the other enhancement figures in this section. In all cases, the output is normalised to have a maximum of one and a minimum of zero.

poor due to the ideal line pattern being violated there, but was improved significantly by pre-processing the image by morphological closing with a small circular structuring element. The closing operation partially fills in the gaps between the dashes, producing a more continuous line. There are also gaps in the response where blobs appear on the lower-left line, and at junctions, because the gradient vectors are no longer pointing in opposite directions here. There is also a slight response from some shoulder edges, in places where these edges have a slight "lip" to them. Despite these problems, the metric produces very little background noise, and a local mean threshold of its response produced an acceptable binary line map which correctly detected most of the lines.

### 5.3.2 Hessian eigenvalues

A technique often used for vessel extraction in the bio-image processing community uses the eigenvalues of the Hessian matrix, which is defined as follows:

$$H = \begin{bmatrix} \frac{\partial^2 I}{\partial x^2} & \frac{\partial^2 I}{\partial xy} \\ \frac{\partial^2 I}{\partial xy} & \frac{\partial^2 I}{\partial y^2} \end{bmatrix} \tag{5.3}$$

and can be thought of as a second order analogue of the gradient vector. A useful property of the Hessian is that its two orthogonal eigenvectors are aligned with the second-order structure of the image at that point, and their corresponding eigenvalues are the second derivatives of $I$ in those directions. A decent mathematical definition of a bright line in 2D is that line pixels should have a small second derivative in one direction (along the line), and a negative second derivative in the orthogonal direction. The elements of the Hessian matrix are computed for every pixel simultaneously by convolving with second order Derivative of Gaussian kernels with standard deviation of

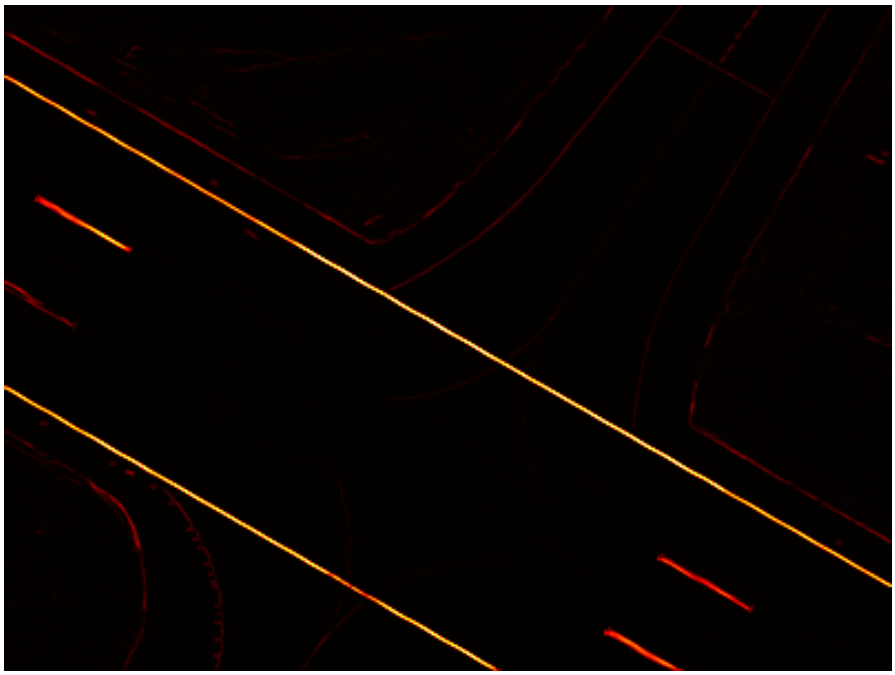


FIGURE 5.5: Output of Hessian eigenvalue metric

one pixel (roughly half the width of the lines we are interested in), and the eigenvalues themselves are calculated explicitly from those elements:

$$\lambda = \frac{H_{xx} + H_{yy}}{2} \pm \frac{\sqrt{(H_{xx} - H_{yy})^2 + 4H_{xy}^2}}{2} \tag{5.4}$$

To turns these eigenvalues into a scalar response, we use Frangi's 2D "vesselness" formula proposed in [43]:

$$e^{-\frac{R^2}{2\beta^2}} \left(1 - e^{-\frac{S^2}{2c^2}}\right) \tag{5.5}$$

where $R$ is the ratio between the smaller magnitude eigenvalue and the larger one, and $S$ is the Frobenius matrix norm of the Hessian, which for a real symmetric $2 \times 2$ matrix is given by

$$S = \sqrt{\lambda_1^2 + \lambda_2^2} \tag{5.6}$$

where $\lambda_1$ and $\lambda_2$ are the matrix eigenvalues, and $\beta$ and $c$ are set to $0.5$ and half the maximum value of $S$ respectively, as recommended in [43].

The output of this metric is shown in Figure 5.5. The main drawback is that it responds not only to lines but also edges - in fact it cannot distinguish between edges and true lines at all, since both will have one small eigenvalue and one negative, with the first eigenvector parallel to the structure and the second one orthogonal. However it performs better than some other algorithms on the uneven lower-left line, because the partial derivatives are calculated using derivative of Gaussian filters, which smooth away the blobs on that line. Another small drawback is that it produces a strong response either side of any sufficiently thin dark line in the image, however we were able to easily correct this by pre-processing the image with a morphological closing to simply remove the dark lines before-hand.

### 5.3.3 Structure tensor

This metric is similar in principal to the gradient vector alignment metric described earlier, in that it measures the degree to which local gradient vectors point along the same line. The structure tensor is defined as follows [51]:

$$J = \begin{bmatrix} (I_x)^2 * g & I_x I_y * g \\ I_x I_y * g & (I_y)^2 * g \end{bmatrix} \tag{5.7}$$

where $I_x$ and $I_y$ are the image's $x$ and $y$ derivatives respectively, and $g$ is a Gaussian kernel. It can be understood as a form of covariance matrix, measuring the correlation between the $x$ and $y$ components of local gradient vectors. In the case of a line, local gradient vectors are mostly parallel or anti-parallel, therefore there is strong correlation between their $x$ and $y$ components.

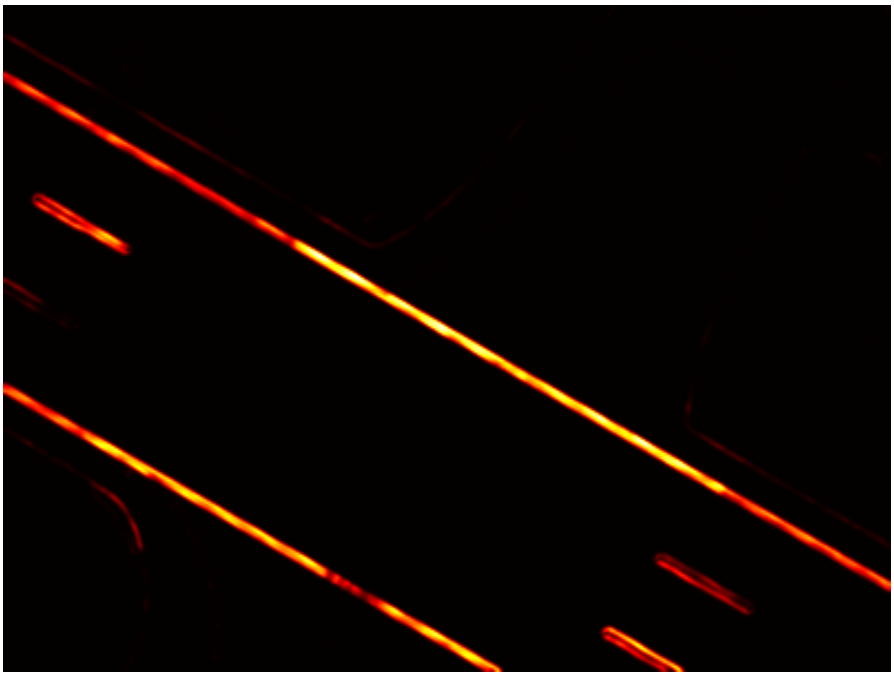


FIGURE 5.6: Output of structure tensor metric

Figure 5.6 shows how this metric's response is similar to that of the first gradient vector alignment metric, except for a much lower response on faint lines, and a higher response on edges. This is because the structure tensor method only requires gradient vectors to lie along the same line to respond strongly, and does not notice whether they are all parallel (as with edges) or anti-parallel (as with lines).

### 5.3.4 Matched filter response

A matched filter approach was implemented, in the manner outlined in [48], in which we convolve the image with a set of eight directional filter with uniformly distributed orientations. These filter have zero derivative along their principle axis, and a Gaussian intensity profile along the perpendicular axis, and are defined as thus:

$$K_n(x, y) = e^{\frac{x\prime^2 + y\prime^2}{2}} \tag{5.8}$$

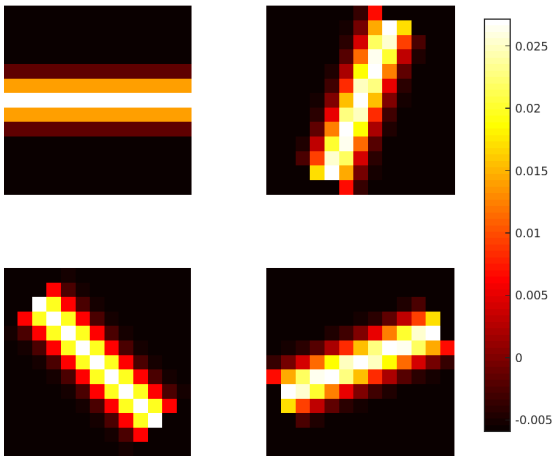



FIGURE 5.7: A subset of the 12 matching filters we used.

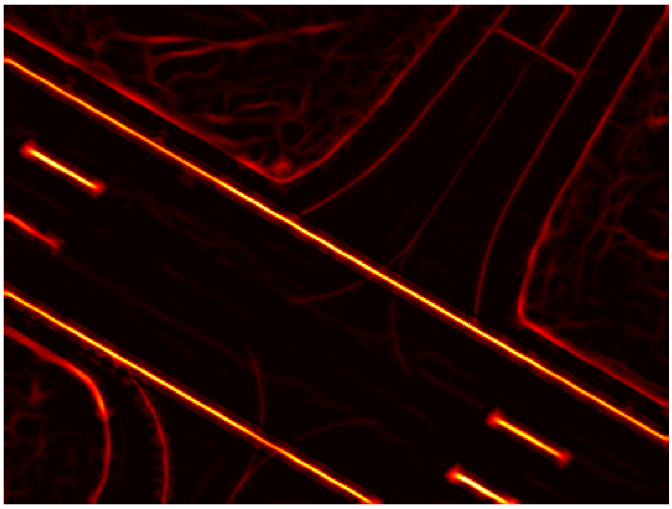
where $x\prime$ and $y\prime$ are the $x$ and $y$ coordinates projected onto an axis with an angle of $\frac{n\pi}{8}$ relative to the $x$-axis, and the standard deviation of the Gaussian profile is one. A graphical representation is given in Figure 5.7. These filter are then normalised to have an integral of zero, otherwise their response strength would be dominated by the local brightness of the image. Each filter should produce a high response at piecewise linear segments of a lines or curves in the image, where the line's width and orientation match that of the filter. To convert the 12 filter responses into a single scalar output for each pixel, we tested three different functions of the filter responses:

1. Take the maximum filter response for each pixel, as detailed in [48].
2. Take the difference between the maximum and mean response for each pixel.
3. Take the reciprocal of the circular variance of the filter responses. Because a filter with orientation $180°$ is the same as one with orientation $0°$, ordinary variance does not make sense for a distribution of angles. Circular variance is an analogue of arithmetic variance for circular quantities which wrap around, such as angles. Circular variance is defined as $1 - R$, where

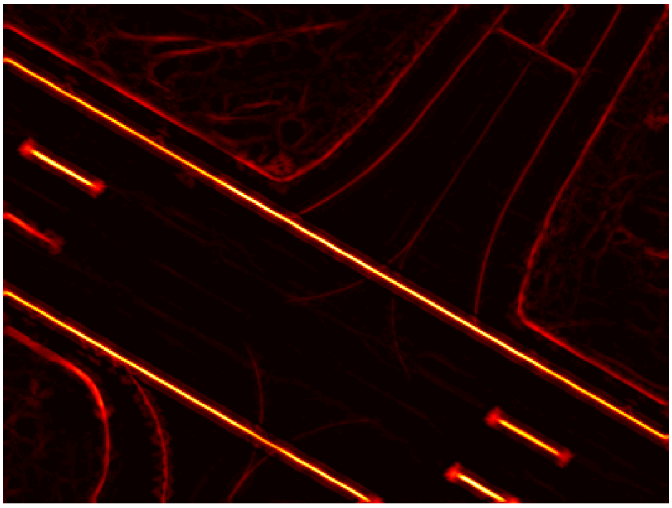
$$R = \frac{1}{N}\sum_{i=1}^{N}(k_i cos(2\pi i/N), k_i sin(2\pi i/N)) \tag{5.9}$$

where $N$ is the number of filters used and $k_i$ is the response of the $i$'th filter. By taking the reciprocal of this quantity we measure the sharpness of the distribution of filter responses over angles, in other words the degree to which the kernels favour one particular direction.

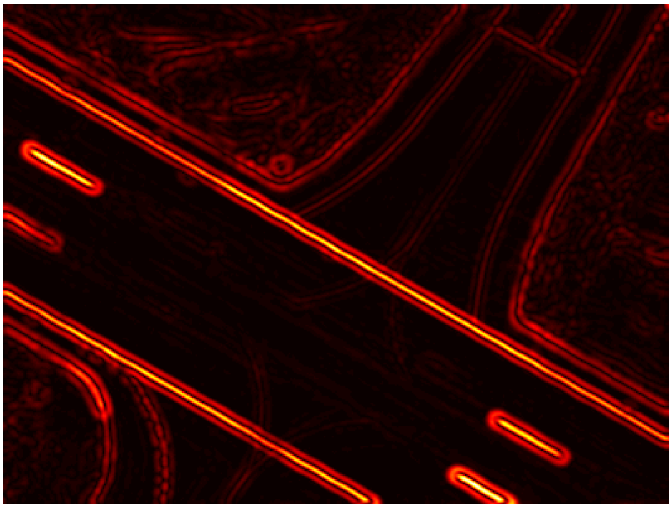
Figure 5.8 shows how this approach produces a smooth response on dashed lines and the lines with blobs, due to the directional blurring effect of the oriented kernels. However, it also has a high response from the background noise in the terrain surrounding the taxiway and runway.



(A) Maximum response



(B) Difference between maximum and mean



(C) Reciprocal of circular variance

FIGURE 5.8: Matched filter response output using different ways of processing the 12 responses into a single value. Maximum response appears to work best.

### 5.3.5 Asymmetry

Since several metrics respond equally strongly to lines and edges, we developed another metric that may help to distinguish between those two features. An edge by definition separates two regions of different intensity, whereas a line should have the same intensity either side of it. To measure the degree to which this symmetry is present, we calculate first derivatives of the image using Derivative of Gaussian (DoG) kernels with a large enough standard deviation that they can sample the intensities either side of a line:

$$G_{x,\sigma}(x,y) = \frac{\partial}{\partial x} G_\sigma(x,y) \tag{5.10}$$

$$G_{y,\sigma}(x,y) = \frac{\partial}{\partial y} G_\sigma(x,y) \tag{5.11}$$

where $G_\sigma(x,y)$ is a normalised Gaussian function in two dimensions. Figure 5.9 shows a graphical representation of $G_{x,\sigma}$ and $G_{y,\sigma}$. Because of the mathematical properties of the convolution operator, these differentiated Gaussian kernels can be used to differentiate the image, with respect to the kernel's orientation and scale space:

$$I_{x,\sigma}(x,y) = I * G_{x,\sigma} \tag{5.12}$$

$$I_{y,\sigma}(x,y) = I * G_{y,\sigma} \tag{5.13}$$

Because DoG kernels are steerable filters [52], we can use the $x$ and $y$ aligned DoG kernels as a basis from which a DoG kernel with any orientation $\mathbf{n} = (cos(\theta), sin(\theta))$ can be calculated as a linear combination:

$$G_{\mathbf{n},\sigma} = cos(\theta) G_{x,\sigma} + sin(\theta) G_{y,\sigma} \tag{5.14}$$

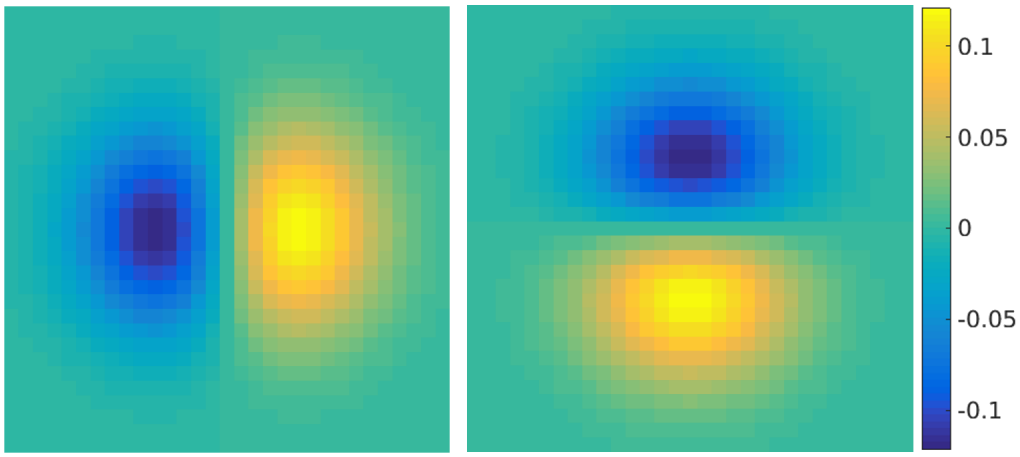
which allows us to differentiate the image in an arbitrary direction $\mathbf{n}$:

$$I_{\mathbf{n},\sigma}(x,y) = cos(\theta)(I * G_{x,\sigma})(x,y) + sin(\theta)(I * G_{y,\sigma})(x,y) \tag{5.15}$$

We then extract the local image orientation at all points by finding the eigenvectors of the Hessian matrix: the one with the small eigenvalue is the local orientation, and the other is perpendicular to it. This allows us to produce a local perpendicular orientation unit vector field:

$$\mathbf{n}(x,y) = \frac{1}{\sqrt{1 + \left(\frac{\lambda_{large}(x,y) - I_{xx}(x,y)}{I_{xy}(x,y)}\right)^2}} \begin{pmatrix} 1 \\ \frac{\lambda_{large}(x,y) - I_{xx}(x,y)}{I_{xy}(x,y)} \end{pmatrix} \tag{5.16}$$

where $\lambda_{large}(x,y)$ is the large magnitude eigenvalue of the Hessian matrix at position $(x,y)$, and $I_{xx}$ and $I_{xy}$ are second image derivatives. Defined in this way, if the point $(x,y)$ is on a line then $\mathbf{n}(x,y)$ will be oriented perpendicular to that line (in the direction of greatest magnitude second derivative). With a perpendicular orientation vector and $x$ and $y$ derivatives (from convolving with $K_x$ and $K_y$) calculated for every point $(x,y)$ in the image, we can now calculate local asymmetry:




(A) $x$ derivative (B) $y$ derivative

FIGURE 5.9: Derivative of Gaussian kernels oriented in the $x$ and $y$ directions, with a standard deviation of five pixels. In both cases the centres of the two lobes are five pixels away from the centre of the kernel. If a line runs between these lobes then the lobes will see the same intensity either side, but if it is an edge then they will see a different intensity.

$$a(x,y) = I_{\mathbf{n},\sigma}(x,y) \tag{5.17}$$

$$= \begin{bmatrix}(I * K_x)(x,y)\\(I * K_y)(x,y)\end{bmatrix} \cdot \mathbf{n}(x,y) \tag{5.18}$$

where $\mathbf{n}$ is a unit vector field in the perpendicular direction. The value of $a$ then describes how asymmetrical the intensity is at each point. Effectively we differentiate the image in a direction perpendicular to the local orientation at every point. Since intensity does change significantly when an edge is crossed but not when a line is crossed, this will in theory return a high value on edges and a low value on lines. We use a standard deviation of five pixels with the DoG kernels to differentiate at a higher scale space than the width of a line, so that we do not find a derivative of zero in the middle of a line.

Figure 5.10 shows how the asymmetry metric has a low response on lines, bordered by a high response either side. Edges on the other hand produce a high response, with no bordering. The response is proportional to the intensity difference either side of the structure. Noisy artefacts can be seen in some places; these are caused by noise in the orientation vector field $\mathbf{n}(x,y)$. This could possibly be reduced by using a multi-scale approach to find $\mathbf{n}(x,y)$, in which the orientation could be computed at multiple scale levels and the strongest orientation selected (strength of orientation can be quantified by the difference between the Hessian's two eigenvalues).

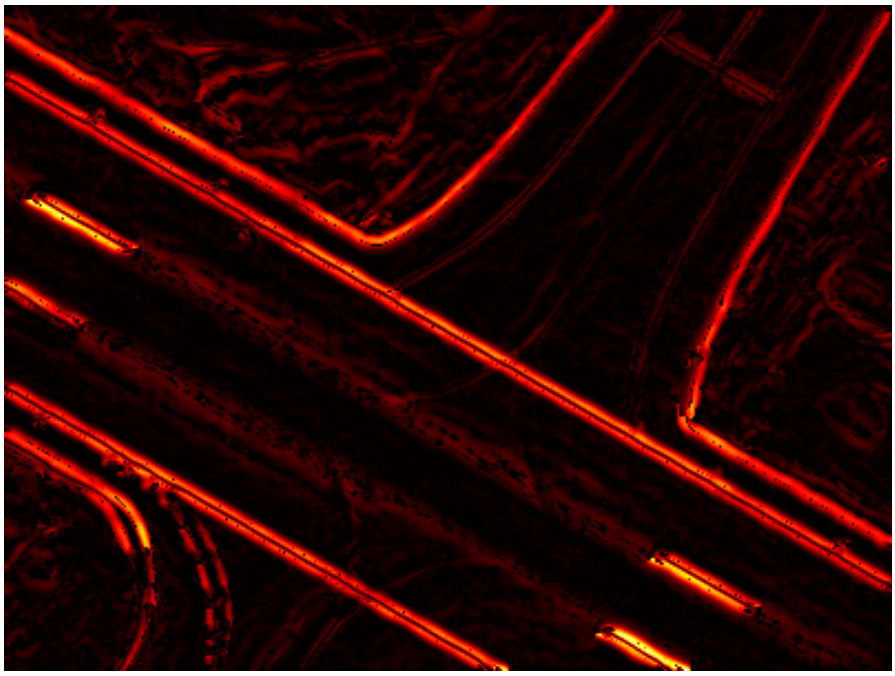


FIGURE 5.10: Output of asymmetry metric

### 5.3.6 Covariance matrix eigenvalues

The covariance matrix of an intensity image is defined as follows:

$$C = \begin{bmatrix} \mu'_{20} & \mu'_{11} \\ \mu'_{11} & \mu'_{02} \end{bmatrix} \tag{5.19}$$

where the $\mu\prime_{ij}$ are central image moments. For an $n \times m$ matrix these are defined as:

$$\mu_{ij} = \sum_{x=1}^{m} \sum_{y=1}^{n} I(x,y)(x-\bar{x})^i (y-\bar{y})^j \tag{5.20}$$

$$\mu\prime_{ij} = \frac{\mu_{ij}}{\mu_{00}} \tag{5.21}$$

where $\bar{x}$ and $\bar{y}$ are the $x$ and $y$ components of the centroid of the image. To compute these values locally within a sliding window, we convolve the image with kernels of the form

$$K_{ij}(x,y) = x^i y^j \tag{5.22}$$

The eigenvectors of the covariance matrix correspond to the principal axes of local second-order structure in the image, and their eigenvalues quantify the correlation of pixel intensities within sliding windows of size dependent on the convolution kernels used. In effect, the pixel intensities within a window are treated as a joint probability density function in $x$ and $y$, and it is this function for which the covariance is calculated. Since the eigenvalues and eigenvectors of the covariance matrix describe the local second order structure of the image in a similar way to those of the Hessian matrix, we use the same vesselness formula as with the Hessian matrix to derive this line metric.

Figure 5.11 shows the output of this metric on our sample region. The triple bands around lines and double bands around edges mean this metric cannot be sufficient on

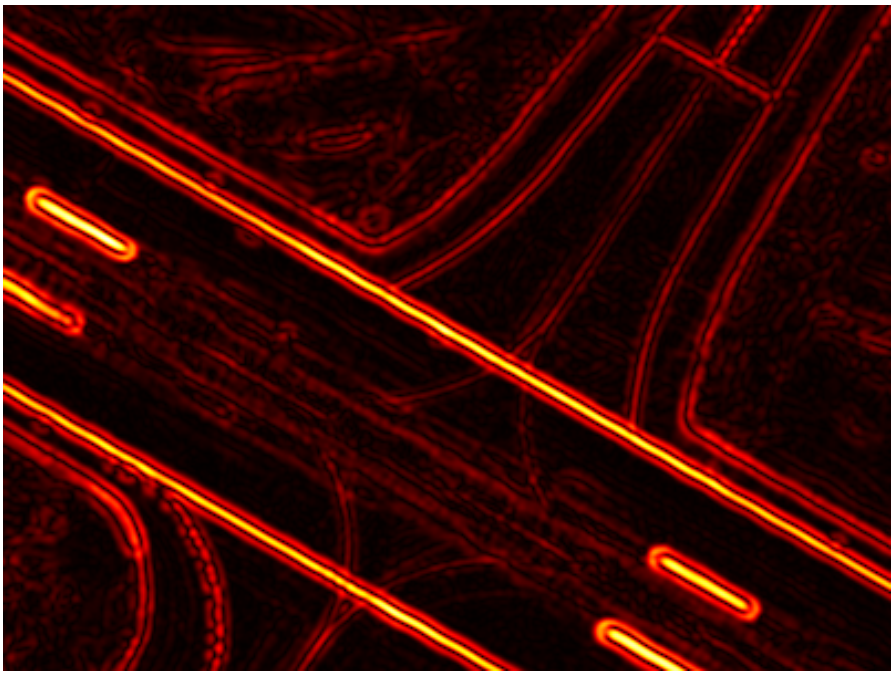


FIGURE 5.11: Output of covariance metric

its own, but it does seem robust against the unevenness of the lines in the top-right and bottom-left of the image.

### 5.3.7 Tophat

The tophat transform is a standard morphological image processing technique [53]. It effectively removes all objects in the image except for those below a certain width. It is computed by subtracting the image's morphological opening from the original image, with the size of the structuring element used in the opening determining the maximum width of the objects that will remain.

Since this metric doesn't rely on derivatives, its response is undiminished at junctions, as shown in Figure 5.12. It is also robust against edges, although it will still respond to edges with an intensity "lip". Its response is equal to the difference between the intensity of the lines and that of their backgrounds, so it responds weakly to faint lines. At no point does it compute the degree to which pixel intensities are arranged along a line, meaning that it will respond just as strongly to small blobs as it does to lines. This also means that it cannot bridge the gaps in the dashed line in the top-right.

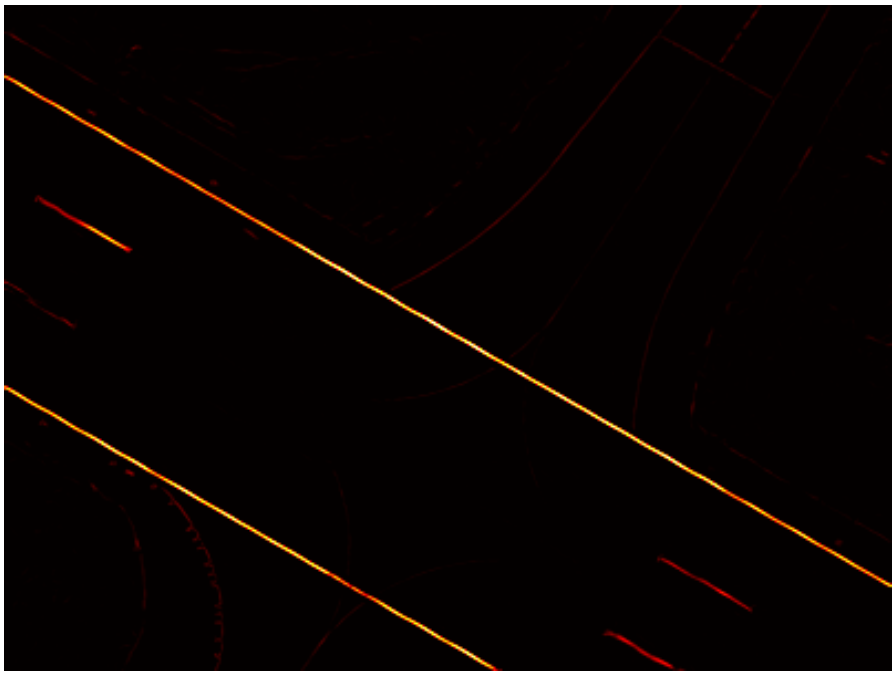


FIGURE 5.12: Output of tophat metric

## 5.4 Machine Learning

Our first approach to producing binary line maps was choose the "best" metric and threshold it with a locally adaptive mean threshold. This approach quickly ran into a number of problems:

- The linear pattern of pixel intensities that most of our metrics screen for doesn't apply at line junctions, endpoints or sharp corners.
- A surprising amount of variation exists in the styles of different lines (see Figure 5.2), and no single metric was able to respond strongly to all different line styles.
- Since many AMDB lines were quite faint and all of the metrics produce a response proportional to contrast, it was difficult to threshold those responses in a way that detects lines while ignoring background noise. Even with locally adaptive thresholding algorithms, gaps still appear in legitimate lines while small patches of background noise were spuriously detected.

Figure 5.13 shows the output of local mean thresholding of the gradient vector alignment metric (see Figure 5.4). Because of the faintness of the central taxiline, it was not possible to detect it without also detecting a lot of background noise. The same problem was found when local thresholding was applied to all other enhancement metrics.

We noticed that different enhancement metrics tended to fail in different cases. Attempts to manually combine different metrics to produce a more robust predictor showed promise, but were unwieldy and difficult to tune correctly. This suggested that a machine learning approach, which could automatically decide how best to combine the metrics using statistical methods, should be the next step. The availability of manually annotated images is a huge asset in this regard, since training data for supervised learning techniques can be derived directly from them.

The task can be framed as a binary classification problem, where each pixel must be classified as line or not-line, based on the seven metrics which are computed for

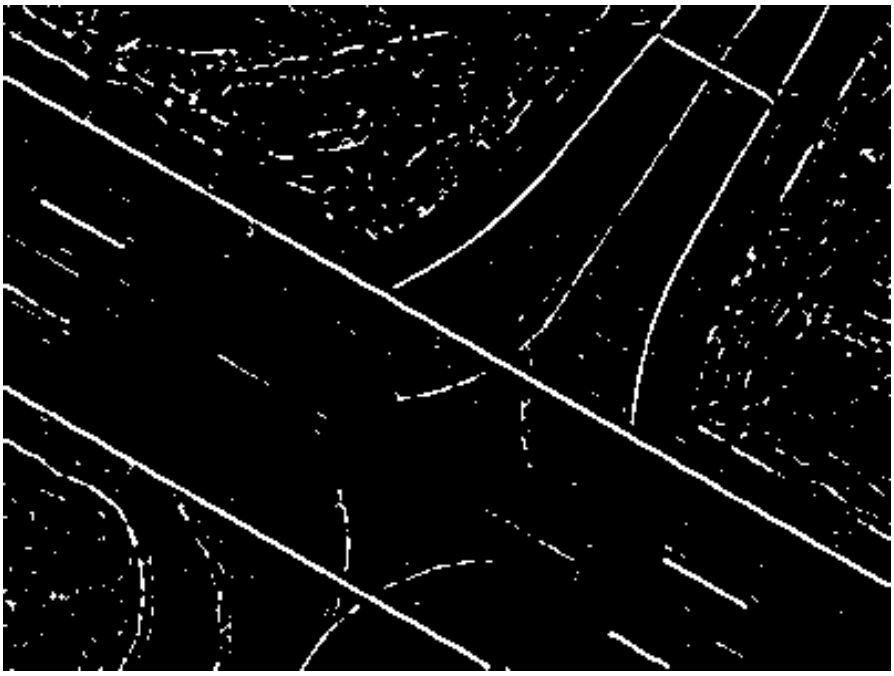


FIGURE 5.13: A binary line map produced by applying local mean thresholding to the gradient vector alignment line metric (Figure 5.4). The threshold at each point is the local mean of the metric within a 5 pixel radius, plus 1% of the maximum metric value - this reduces over-detection in noisy background areas.

each pixel. We also augmented this feature set by including the local mean of colour saturation for each pixel. So far we have worked with greyscale data, for simplicity and because most of the techniques we have applied were developed for greyscale data, but RGB information is available in our dataset. We therefore provided pixel saturation to the classifier, since the lines we are interested in occur in runways, taxiways and aprons, which tend to be less colourful than many other parts of the image, such as vegetation.

We have begun by applying a Naive Bayesian classifier, one of the simplest supervised learning algorithms [54]. We trained the classifier using our seven metrics, plus the local mean of the image saturation (found by convolving the saturation channel of the HSV image with a Gaussian kernel) as an eighth metric. This improved performance significantly, since low saturation is strongly correlated with the presence of airport lines. The training set was produced by first overlaying the line GIS objects onto the old image, to identify which pixels in that image belong to airport lines. To convert the GIS line segments, which are vector objects stored as geographic coordinates, we first map them to image coordinates using the image's embedded georeference data. We then pass those coordinate pairs to a freely available MATLAB implementation of Bresenham's line algorithm [55], which rasterizes a pair of points describing a line segment into a binary image. The union of the output of Bresenham's algorithm for all line segments in the GIS data identifies all AMDB line pixels in the image, from which we extract our training set by selecting 2500 line pixels are 2500 non-line pixels at random. This gives us an array of 5000 8-dimensional vectors each corresponding to a positive or negative example.

The output classification when applied to our sample region is showing in Figure 5.14. In Figure 5.15, we show the classifier's posterior probability level as a heatmap.

We also show, in Figure 5.16, correlation coefficients which describe how relevant the classifier found each of our features be for predicting the class. The correlation coefficient between two features $i$ and $j$ is defined as

$$R(i,j) = \frac{C(i,j)}{\sqrt{C(i,i)C(j,j)}} \tag{5.23}$$

where $C(i,j)$ is the covariance between features $i$ and $j$, and $C(i,i)$ is the variance of feature $i$. Correlation can be thought of as normalised covariance, which is invariant to the absolute scales of the features, and therefore guaranteed to lie in the interval $[-1,1]$. Figure 5.15 shows the correlation between each of the reviewed line enhancement features and the ground truth line classification (which is $1$ or $0$ depending on the AMDB). This gives us a quantitative measure of how effective each of our line enhancement algorithms are as predictors of whether or not a given pixel belongs to an AMDB line. Note that we have shown absolute values for the correlation coefficients - the coefficient for asymmetry is in fact negative, since AMDB lines tend to have a symmetrical intensity profile.

It can be seen the tophat transform is the most salient feature, while the structure tensor is the least. The poor performance of the structure tensor can be attributed to its high sensitivity to step edges, and disproportionately low response to faint lines such as taxilines. Tophat meanwhile will only respond to features that are erased by a morphological opening, and must therefore have a thickness below that of the structuring element used. Step edges are mostly unaffected by openings, and so do not appear in the tophat transform.

As expected, the correlations of all the individual features are all quite low (none greater than $0.35$), indicating that none of them are sufficient on their own to predict if a given pixel belongs to a line or not. However, Figure 5.14 shows that the confidence of the naive Bayes classifier (expressed as the posterior probability of the class label) is correctly above $50\%$ in many places, showing that statistical approaches can indeed be used to combine multiple features into a useful probability score. Unfortunately the posterior probability is still very low for the faint central taxiline (see Figure 5.15), which suggests that the features used are not powerful enough to detect all AMDB lines. This is likely due to contrast sensitivity, since all of the features used respond more strongly to objects with greater contrast, and the taxiline concerned was very faint.

## 5.5 Conclusion

Although a wealth of literature exists concerning curvilinear feature extraction in bio-medical image data, we found there were significant challenges in applying these techniques to airport images. Due to the complexity of the images in our dataset it is difficult to find a metric which responds well to lines in all the forms they appear in, but does not produce false positives at edges or background noise. It is likely that many approaches developed for bio-medical applications would need to be adapted significantly to work on airports, since our images contain a huge variety of patterns which are not present in the images for which those approaches were developed. For example, retinal images tend not to contain dashed lines like those typically found in stopbars (see Figure 5.2(c)). An airport line detector must be robust against such variations in style. Even if the line detector is perfect, in that it detects lines conforming to patterns that appear in airports and rejects other features such as edges, the images are large

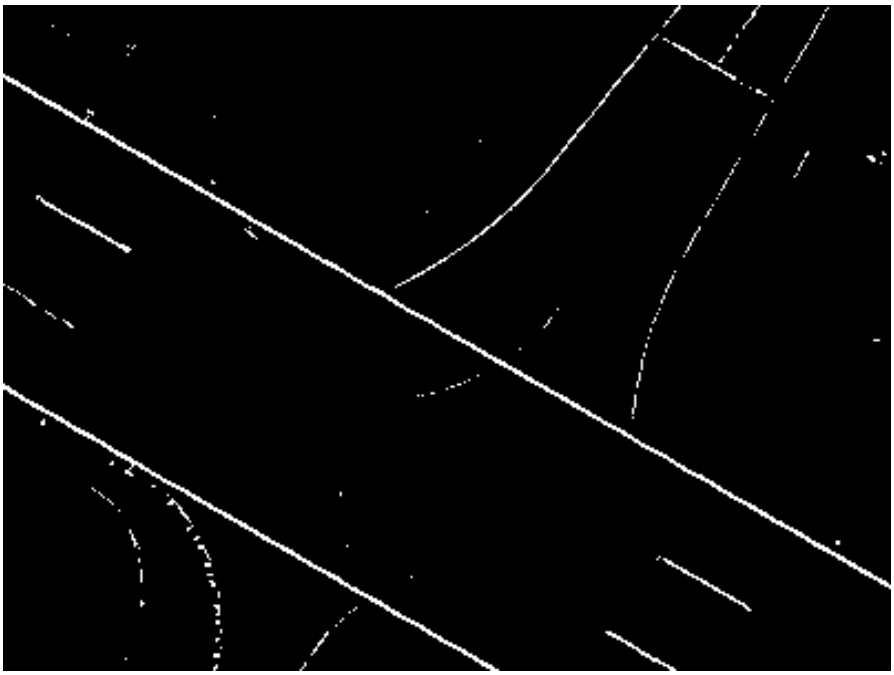


FIGURE 5.14: Output of the Naive Bayesian classifier on our sample region.

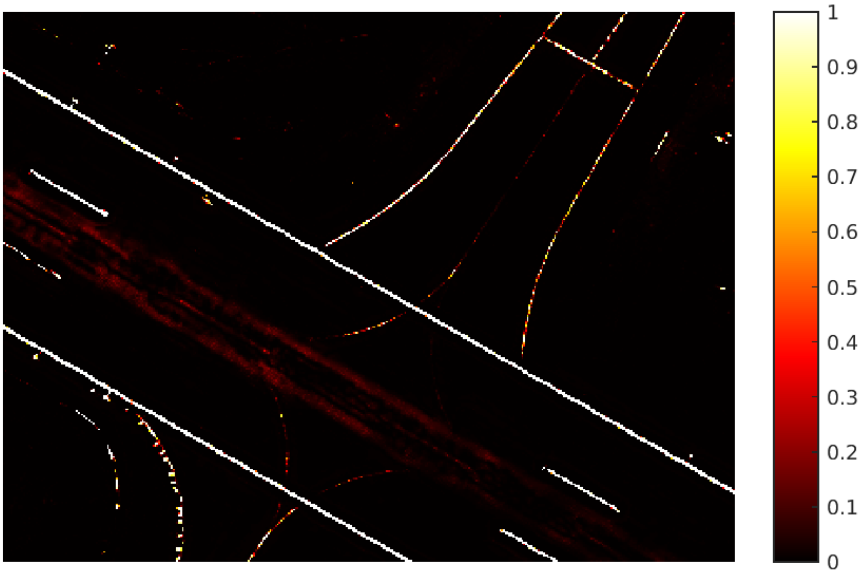



FIGURE 5.15: Posterior probability of the Naive Bayesian classifier on our sample region. Very low confidence can be seen on the faint central taxiline; more salient enhancement metrics and/or a more powerful classifier would be needed to detect this feature.

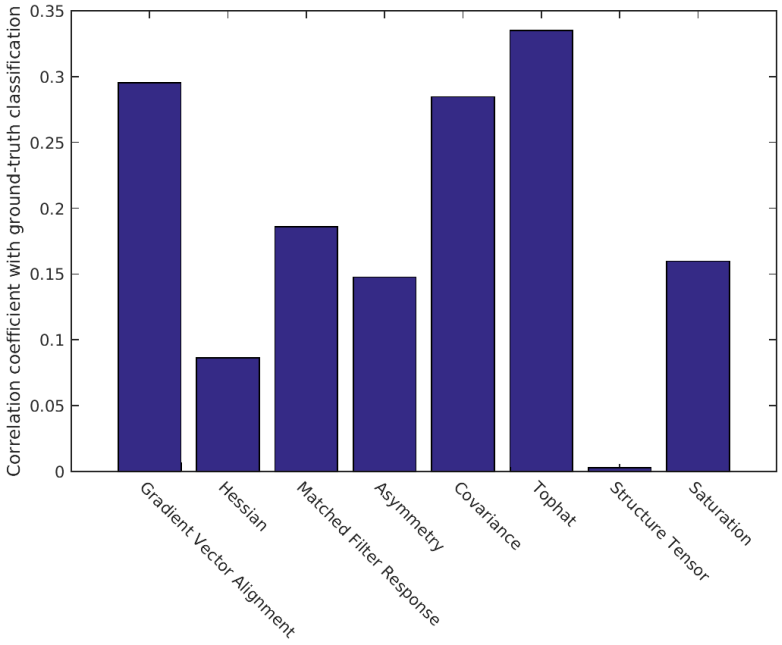



FIGURE 5.16: Correlation coefficients between the training features and the ground truth class.

enough that those patterns will occur in some places outside of the airports and will be detected in those places. However, the majority of those can probably be filtered out based on their distance from the airport (see Figure 4.19), which can be defined using pre-existing GIS data or by detected runways.

This chapter has only addressed the problem of detecting lines, and has not dealt with how to detect *changes* to lines. An ideal system would first detect lines in both images, and then decide where lines had been added or removed from the airport and report such changes to a human operator for verification. A potential solution to this problem is outlined in Figure 5.17. The left branch represents a post-classification pixel-based change detection approach, while the right is object based.

Future research in this area may benefit from pre-processing techniques which enhance the contrast and uniformity of the lines. We note that in most airports we have encountered, the airport lines are in fact painted in yellow, not white. This suggests that they could be enhanced by applying a colour deconvolution, as outlined in [56]. Doing so could improve the lines' contrast greatly, making lines which appear very faint in greyscale to stand out much more by exploiting their difference in colour against their background, which is ignored in this chapter. Another pre-processing technique which may make line detection easier is anisotropic diffusion, as introduced in [57] and applied in [51]. Anisotropic diffusion blurs an image whilst preserving the edges that separate different regions, by applying directional blur which is sensitive to local image orientation. Deguchi *et al.* [51] use it to smooth lines of text in the direction of their principal axis, which has the effect of smoothing the letters in a line into a streak, without merging close parallel lines into one or reducing the intensity of the lines, as would happen with a uniform Gaussian blur. The same approach could be applied to our images to blur lines such as that stopbar in Figure 5.2(c) into a smooth, unbroken line, which would be easier to detect. These pre-processing techniques will benefit any line detection approach, not just the ones tested in this chapter.

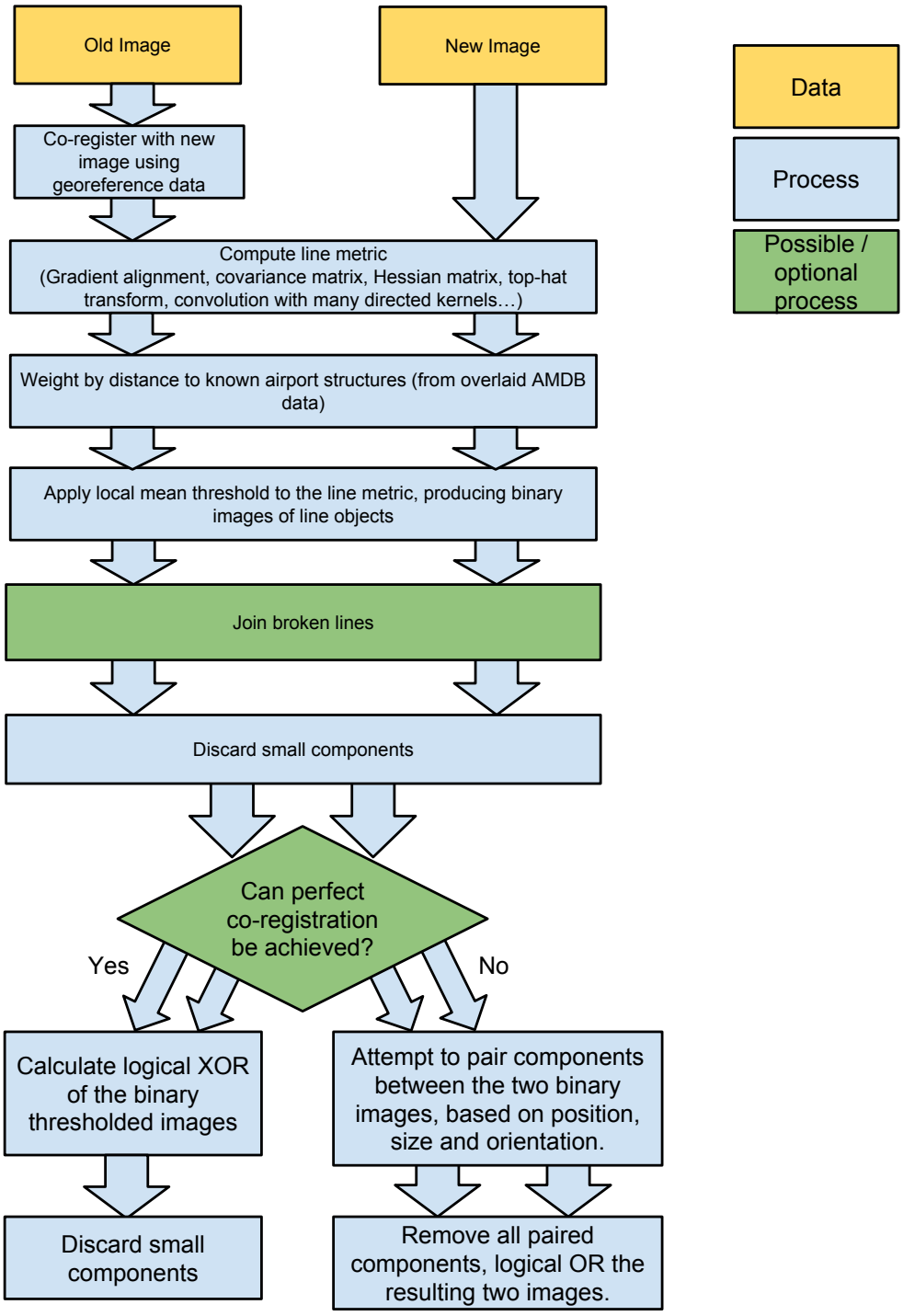



FIGURE 5.17: A workflow that could potentially be used for detecting the addition or removal of lines in a bi-temporal image pair.

# Chapter 6

# Line Growing

## 6.1 Introduction

Although the results in the previous chapter showed promise, they suffered from a fundamental limitation. Because the line enhancement methods treated all parts of the image equally and reported lines wherever they found them, they would still require a human operator to decide which lines were relevant to the AMDB and which weren't. Furthermore, we anticipate a difficulty with the post classification change detection as was planned in the previous chapter. Some of the lines we wish to detect are very faint, or difficult to detect in some localised places for other reasons (such as junctions or unusual line style). In these cases, gaps are likely to appear in the generated binary line map, and these gaps are likely to vary between the old and new images due to noise, since the response from the line enhancement metrics is likely to be very close to the detection threshold in those places. These variations could produce false positives in a change detection algorithm.

Our solution to these problems exploits both prior knowledge of how airports are structured and our ability to detect runways, to detect only lines which are relevant to the AMDB. Our prior knowledge relies on only one fact: since AMDB lines exist to guide taxiing aircraft around the airport, it must be possible to reach a runway from any point on an AMDB line without leaving the line network. The only exception to this rule would be a taxiline connecting two aprons. To deal with the issue of gaps in the binary line map, we developed a technique which bridges these gaps by extrapolating lines at their endpoints and connecting them to other lines which appear to continue down the same path. This work was greatly aided by our discovery of a very effective algorithm for line detection, for which we also found a freely available Java implementation which we were able to call from within MATLAB.

## 6.2 Steger's Line Detection Algorithm

Detection of lines, which are long thin objects with a roughly Gaussian intensity profile, is a separate problem to that of detecting edges, which separate bright and dark regions and have a roughly Heaviside shaped intensity profile. An algorithm (which we shall refer to as Steger's algorithm) for line detection was presented by Steger in [58], as the result of a detailed mathematical analysis of what lines are and how they respond to various convolution kernels. Briefly, this algorithm works by first detecting sub-pixel line points, then linking them together to form complete lines. The line points are found by constructing a local quadratic model of intensity in the neighbourhood of each pixel (neighbourhood size being dependent on a scale parameter), and finding the stationary point of this function along the line which intersects the centre of the pixel and is parallel to the line's normal vector. This normal vector is computed for each

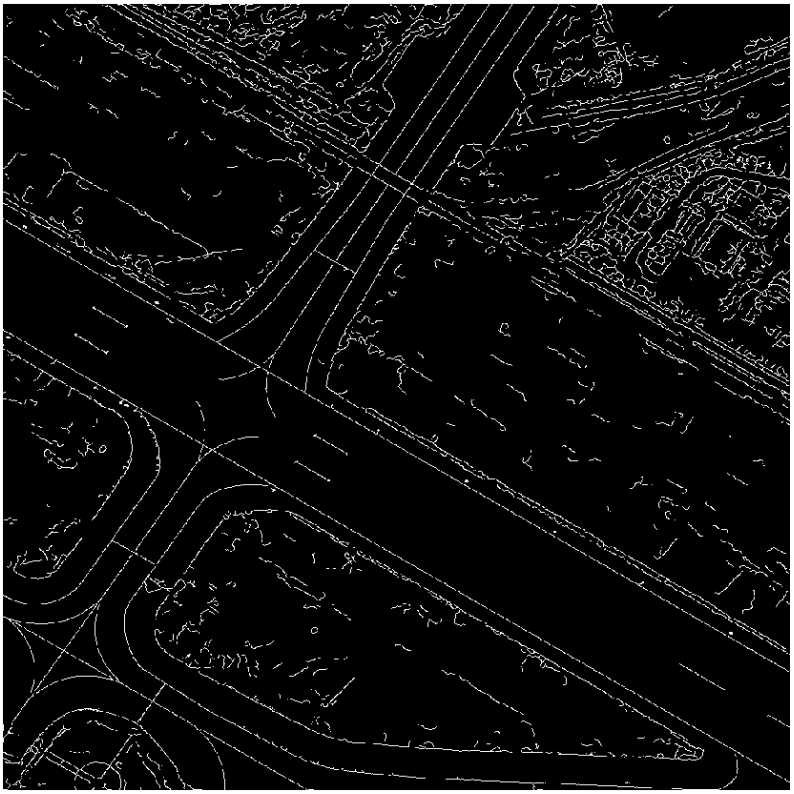
pixel by finding the largest eigenvector of the Hessian matrix (that is, the eigenvector with the largest magnitude eigenvalue). The parameters of the quadratic model, meanwhile, are found by estimating the image's first and second partial derivatives by convolving with derivative of Gaussian kernels; the model is then computed as a second order Taylor expansion about the pixel's centre using these derivatives. The convolutions required to compute the Hessian matrix are already performed in estimating the model parameters, therefore the five convolutions (for the two first and three second derivatives) form the bulk of the computational expense of the algorithm. As described so far, the process will find line points for every pixel, but of course not all pixels should be classified as belonging to a line - Steger's algorithm therefore classifies pixels as belonging to a line only if the line point is found to lie within the boundaries of the pixel, otherwise it is ignored and the line point is discarded. Once line points have been found, they are linked together to form lines, in a process that uses two user specified thresholds for the perpendicular second derivative to decide when to stop adding points.

A recently written, open source Java implementation of Steger's algorithm can be found at [59], originally written as a plugin for the biomedical image analysis packages ImageJ and Fiji. After producing a simple MATLAB wrapper for the Java classes and applying it to our images, we found that it performs quite robustly, although it does require three user specified parameters. The original algorithm requires standard deviation for the DoG convolution kernels, and an upper and lower threshold for the perpendicular second derivative. The Java implementation takes three different, more user-friendly parameters, which it then maps to the original three: line width, upper contrast and lower contrast. We estimated these parameters by manual image inspection and a small amount of trial-and-error. Despite these user parameters we still consider Steger's algorithm to be very useful, especially since these parameters have a clear intuitive meaning and did not require large amounts of trial-and-error. The line width and contrast parameters used throughout this chapter are

- width: 2
- low contrast: 1
- high contrast: 60

These parameters make the algorithm very sensitive, so it detects many false positives but also very few false negatives (see Figure 6.1). This is deliberate, and the rest of this chapter details how we remove these false positives, leaving only the lines which are relevant to the AMDB.

(A) Original image



(B) Ridges detected by Steger's algorithm

FIGURE 6.1: Binarised output of Steger's algorithm on a small region of Bahrain airport. Steger's algorithm returns piecewise linear curves composed of sub-pixel line points, which we round to integer values to produce this binarised output.

## 6.3 Method

### 6.3.1 Overview

We begin with a concise summary of the process. The first step is to run the runway detection algorithm described in Chapter 2, to obtain polygons representing the runways in the image. Manually annotated GIS polygons can be used if available, although in our experiments we used the runway transform. Next, run Steger's algorithm to generate a binary line map such as the one in Figure 6.1, and split it into simple curves with no branches by removing branch points. We can now detect the exit lines in the image by finding curve segments that touch the runway polygons. These curve segments then act as seeds, which we grow into the whole taxiline network by recursively joining them at their endpoints to other curve segments in the image. Curve segments which can be reached by following a runway's exit lines in this way are kept, and all other curves are discarded. The process by which we join the separated curve segments together also bridges small gaps that appear in the lines due to low contrast, junctions or taxiway signs, and will be elaborated on later.

What follows is a more in depth explanation of the process. A small ($1000 \times 1000$ pixels) subregion of Bahrain airport was used to develop this technique and is used to explain it in this section.

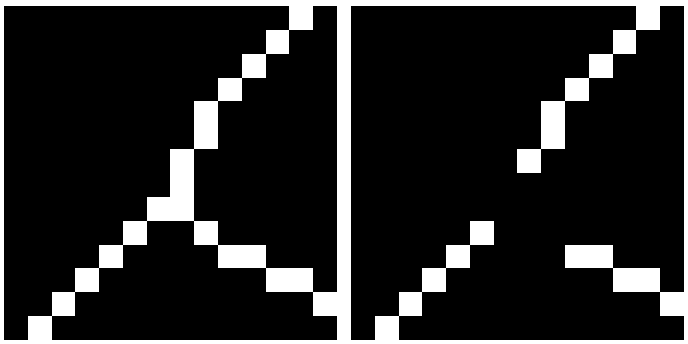
### 6.3.2 Detailed explanation

#### Line detection and pre-processing

We begin by running Steger's algorithm on our image, with parameters as described in section 6.2. We chose to binarise the data (as shown in Figure 6.1) instead of directly using the sub-pixel line points returned by Steger's algorithm, since that allowed us to use MATLAB's many built in binary morphological functions, allowing for more rapid prototyping. We then pre-process this data by removing "messy" connected components based on mean branch length. The idea is to remove connected components that can be safely dismissed as irrelevant based on their shortness and curvature, since AMDB lines tend to be long and smooth. While it is not possible to remove all irrelevant lines this way (since relevant lines can sometimes be broken into short, messy segments), removing obvious duds makes it less likely that an AMDB line will join to such a dud, and will at the very least improve the running time of subsequent steps. As a messiness heuristic, we estimate the mean branch length of each connected component as the number of pixels in that component divided by the number of branch points plus one, and remove all components with a below average value of that heuristic.

Next, we separate the line mask into single-branched line segments by punching a hole at every branchpoint, as demonstrated in Figure 6.2. These branchpoints are detected as single pixels with MATLAB's built in `bwmorph` function, and the holes are made by removing pixels in a 2-pixel radius of each one.

As a further pre-processing step we also remove line segments shorter than 50 pixels with a tortuosity greater than an empirically chosen threshold of 1.2. The tortuosity of a line is defined as the ratio between the line's length and the Euclidean distance between its endpoints, so in this way we remove short lines with high curvature - since AMDB lines tend to be locally very straight and smooth. The final result of this preprocessing is shown in Figure 6.3.



(A) Original (B) Separated

FIGURE 6.2: Separating the line mask into separate curves by removing pixels in the vicinity of branchpoints.

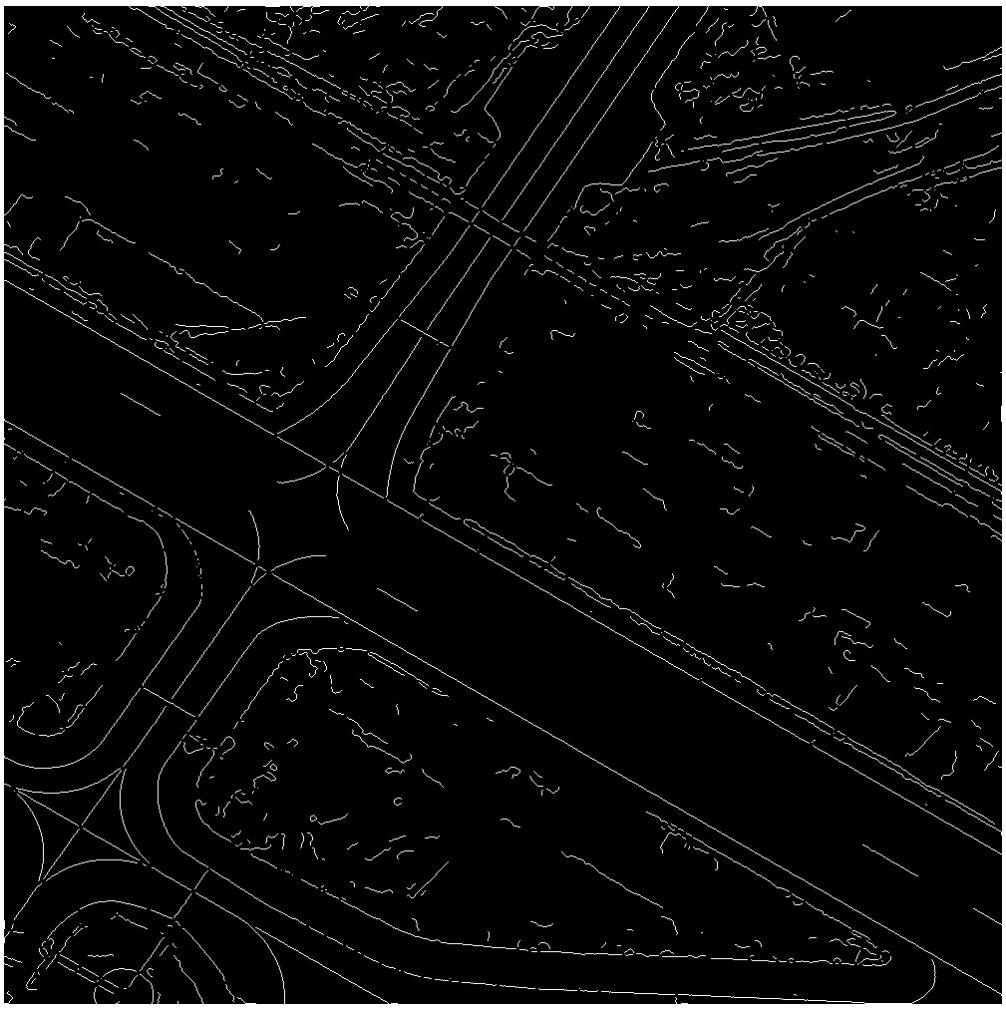


FIGURE 6.3: Output of filtering and splitting operations.

### Seed selection

We now seed our taxilines by detecting the line segments which correspond to exit lines. This is accomplished by first acquiring a binary mask of the runway areas (either by GIS data or by runway transform), then morphologically eroding that mask with a circular structuring element. We then perform morphological reconstruction with the eroded runways as the marker and the pre-processed lines as the mask. In the case of binary data, morphological reconstruction (as described in [60]) will remove from the mask all connected components which do not intersect with any connected components in the marker, where the marker and mask are binary images of equal dimensions. In our case this yields a new binary mask in which only curves which penetrate into the runways are retained. We then filter these remaining curves by removing any which are parallel to the runway (within a tolerance of $30^\circ$) - this removes curves caused by runway markings or other spurious sources. What remains (see Figure 6.4) are provisionally assumed to be exit lines.

FIGURE 6.4: Exit lines selected as seeds; these will be grown by connecting them to other curves (in white) which continue them.

### Bridging

The main part of the algorithm commences now, in which we start with the seed curves (exit lines), and join them recursively to other curves by their endpoints to find the rest of the taxiline network. The idea is that since we assume a priori that the seeds are AMDB lines, any curves that appear to continue them are probably also AMDB lines.

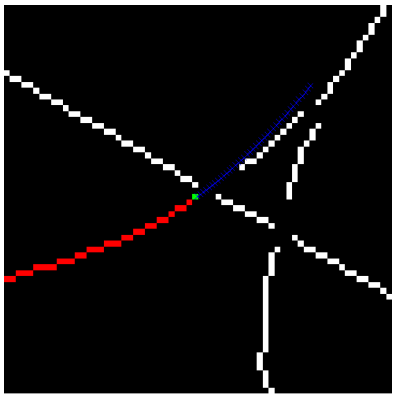


FIGURE 6.5: In red: the line to be extended. In white: candidate curves that our line may be joined to. In green: the endpoint by which the red line will be joined to another curve. In blue: points on the red curve's predicted path at which the distance field is sampled.

These continuing curves are connected to the known AMDB lines until no more valid connections can be found.

To connect a growing AMDB line to a curve that continues it, we must look in the vicinity of its endpoints for candidate curves. However it is not sufficient to simply find any other curve with an endpoint in this vicinity, nor can we require that a continuing curve have an endpoint in this vicinity in order for our line to join it. Our approach is to extrapolate the growing line at its endpoint, and any curve that the extrapolated part comes sufficiently close to is then considered as a connection candidate. The extrapolation is performed by taking the twenty pixels leading up to the endpoint in question and fitting a quadratic model to them - that is we assume the $x$ and $y$ coordinates of those pixels are quadratic functions of their index when sorted by geodesic distance to the endpoint. The coefficients of these two quadratic models are estimated by least square error fitting, using MATLAB's built in `mldivide` function.

Connection candidates are found by first generating a Euclidean distance field, which stores the distance to the nearest line pixel (and the index of that nearest pixel) for every pixel in the image. Figure 6.5 shows how this distance field is sampled at a discrete set of points along the predicted path of the curve to be extended. This yields a 1D signal, the minima of which occur at sampling points where the curve's predicted path passes close to another curve that may be joined. These minima therefore represent potential connection candidates, and the issue now is how exactly to choose which connection (if any) to make. For now, we restrict a growing taxiline to only join a single curve at a time - the case of branching taxilines is handled in the next subsection. Connection candidates must pass two requirements to be used: the extrapolated curve must pass within three pixels of the connection point, and the direction of the curve to be joined on to must (at the connection point) be within 30° of that of the growing curve. This exploits the prior knowledge that planes do not make sharp turns while taxiing, therefore it must be possible to follow taxilines back to a runway while only making smooth turns.

When multiple connection candidates are present, we resolve the conflict by choosing the first one. If no qualifying connections are found then the endpoint being considered for extension is labelled a dud and will not be considered again. If a qualifying connection is found, then the endpoint of the growing line is joined to the closest point on the candidate curve to the extrapolated path.

### Branching

It is not uncommon for taxilines to split into multiple branches (see Figure 6.6). Rather than account for this by letting one line join to multiple connection points (which can be difficult when different connection points do not produce distinct minima in the distance signal), we allow curves with endpoints close to the established taxiline to join onto the established taxiline, subject to the same criteria outlined in the previous subsection. To detect nearby curves which may join onto the taxiline, we perform morphological dilation of the existing taxiline, followed by morphological reconstruction in which the dilated taxiline is used as a marker and the split curves image (i.e. Figure 6.3) as the mask. A disk shaped structuring element is used for dilation; the radius of this disk determines the maximum distance a curve may be from the existing taxiline for it to be considered as a branch candidate - we use a radius of five pixels.

Once a curve has been captured in such a way, we continue to grow it at its other endpoint just as we did with the first taxiline. This also provides a way for us to filter out non-taxiline curves that will sometimes be captured in this method; if a captured curve does not grow to a significant length, we discard it. We chose 200 pixels is used as an empirical threshold, but this parameter is not very sensitive since taxiline branches tend to be far longer than this. We perform the branching step once the current taxiline has finished growing, i.e. none of its endpoints can form valid connections to other curves. If no new branches were kept after this step then the algorithm terminates, otherwise the branching step repeats, since the new branches may themselves have more branches to find.

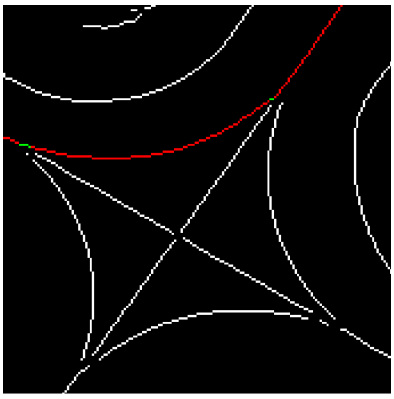


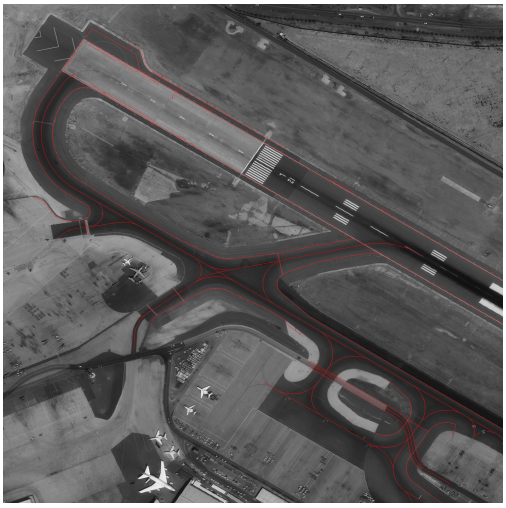
FIGURE 6.6: A taxiway intersection where the taxilines branch. In red: the current taxiline. In green: pixels where gaps in the original line data were bridged as the taxiline grew. Since the captured taxiline has not branched, the other taxiline branches must be added by joining them onto the captured taxiline.

## 6.4 Results

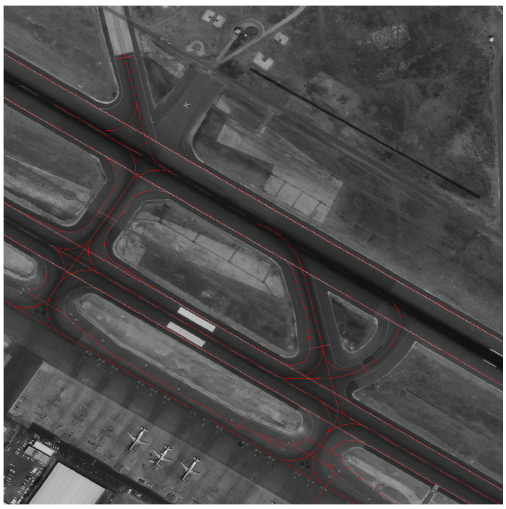
Our results so far have been encouraging. Although the algorithm has only been tested on Bahrain airport due to time constraints, we find that at least in this case the taxilines, stopbars and exit lines are detected correctly with no false negatives. However, some false positives are present. The most common culprits of these false positives are roads crossing perpendicular to the taxiways. These roads are delimited by white lines on either side, which are picked up by the algorithm and followed for as far as they remain visible. It is difficult to filter out road markings, because morphologically speaking they are very similar to stopbars, which we do wish to detect - both are bright lines crossing perpendicular to taxiways, and the road itself does not always have a different shade or texture to the taxiway. However, it should be possible in theory to distinguish them by the fact that they continue beyond the boundaries of the taxiway.

Another noteworthy observation is the fact that not all of the painted lines detected have a corresponding entry in the AMDB. In Figure 6.7a it can be seen that three parallel lines were detected along the northern taxiway. These lines all have similar appearance and are all contained within the same taxiway, but only the centreline, stopbar and exit lines have entries in the AMDB - the two outer lines that delimit the taxiway's shoulders have no corresponding shapefile. They are detected in the north taxiway but not others because of the road crossing the north taxiway, which joins the outer shoulder lines to the centreline and ultimately to the exit lines.

(A)



(B)



(C)

FIGURE 6.7: Output of the line growing algorithm on three subregions of Bahrain International Airport.

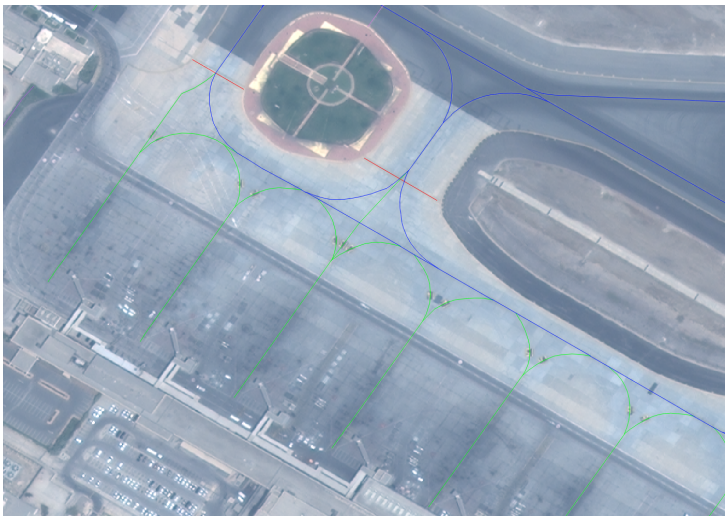
## 6.5 Conclusion

In this chapter, we have trialled an alternative approach to detecting line type AMDB features. Firstly, rather than applying line enhancement to an entire image and attempting to threshold the response, as in the previous chapter, we use a more mature and reliable ridge detection algorithm which existed for many years. Secondly, in order to filter the resulting output to detect only AMDB lines, we exploit the fact that taxilines must eventually join onto a runway. Since we know where the runways are (either by manually annotated GIS or by automatic runway detection), we can discriminate between taxilines and irrelevant lines based on whether or not they can be followed back to a runway.

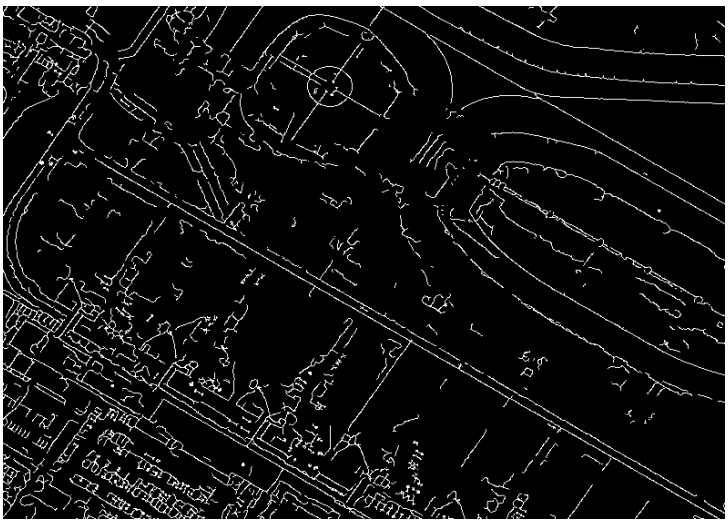
The results shown in this section are a proof of concept, and can probably be improved on greatly by implementing the basic idea in a different way. The current implementation has been hobbled somewhat by the use of a binary line map instead of the sub-pixel network of linked line points that Steger's algorithm outputs. This decision was made to allow faster prototyping under limited time, but using the line points instead of binary data will most likely be both more accurate and run faster. Another opportunity to improve results would be to pre-process the image with one or possibly several of the line enhancement methods examined in the previous chapter. This could improve the results of Steger's algorithm by accentuating lines and suppressing other features, resulting in fewer breaks in detected lines and fewer false positives in noisy background areas. However, we note that markings in airport aprons will probably remain difficult to detect due to the inhomogeneous background, presence of many aircraft and surface vehicles, and the faintness of the lines in those regions (see Figure 6.8).

From a change detection point of view, discrete line objects like the ones we extract here are easier to compare between two images than per-pixel metrics such as those explored in the previous chapter. In order to accurately perform pixel-wise change detection, two images must first be co-registered, preferably to sub-pixel accuracy. Connected components of lines, however, can be matched together between two images based on morphological and spatial properties (e.g. number of branches, orientation, number and degree of intersections). Once the same group of lines has been detected in both images and matched together, they can be compared to see if and where branches have been added or removed. Any branch in the new image that cannot be matched to one in the old image in this way can be considered new.

(A) Apron



(B) AMDB lines



(C) Steger lines

FIGURE 6.8: (a) An airport apron. (b) GIS data overlaid, showing taxilines (blue), stopbars (red) and standing lines (green). (c) The Steger's algorithm output for that area; due to the high complexity of the region and faintness of the lines, the output is not usable.

# Chapter 7

# Conclusion

AMDB update is a very broad problem with many parts, each of which can be solved in many ways. Most of our best results have been obtained by focussing on a single sub-problem and breaking that down into a series of standard image processing tasks, often exploiting prior knowledge along the way. For example, our runway detection approach in Chapter 2 exploits the fact that runways are known to have runway markings that lie along the same straight line, have long side-stripes, and are terminated by threshold marking stripes. We detect these features using Canny edge detection, standard Hough Transform and Fourier analysis, respectively. Likewise, our line growing work in Chapter 6 assumes that taxilines must always join onto runways eventually, and detects curvilinear structures using the well established Steger algorithm. This yielded more useful results than our efforts in Chapter 5, since the task was reduced to joining line segments together, rather than attempting to detect all lines in the image by curvilinear enhancement techniques.

Prior knowledge has played a large role in many of our results, partly because the problems we attempt to solve cannot all be adequately described mathematically. For example, blood vessels in retinal imagery can be described mathematically as intensity ridges in the image, whereas runways are much more complex objects. We found that most of the existing literature in runway detection (such as [7]) also exploits prior knowledge heavily. Exploiting prior knowledge where possible allows one to simplify complex problems.

Our main contributions to Jeppesen's effort to automate the AMDB update process are as follows. In Chapter 2 we present a method for detecting and segmenting runways, which is robust and accurate on the provided dataset. Chapter 3 investigated image registration techniques, while Chpater 4 used that registration as a precursor for direct change detection. Chapter 5 tests a number of curvilinear enhancement algorithms to help detect line type AMDB features such as taxilines and stopbars. Finally, Chapter 6 exploits prior knowledge of airport structure to detect relevant AMDB lines and ignore irrelevant lines.

Our overall assessment is that the visual complexity of most AMDB features, the variety of appearance within a single class of feature, and the overwhelming abundance of non-AMDB objects makes a fully reliable automatic solution very difficult to achieve. For that reason we recommend investigating semi-automated approaches, which have the potential to save great amounts of time while being far more attainable. Most of the work in this thesis can integrated into a semi-automated framework. For example, runways and taxilines can be detected and segmented automatically with humans only checking the algorithm and correcting it when necessary. Accurate image registration, manual or otherwise, allows for images such as those in Figure 3.2 to be produced, which can help a human operator to rapidly locate added or removed objects. The methods explored in Chapter 4 meanwhile were developed explicitly to alert

a human operator to areas of significant change at a coarse scale, and could be adapted to work at the finer scales seen in Figure 3.2. Finally, many more semi-automatic methods exist in the literature which have not been investigated here. For example, active contour models, first proposed in [61], are capable of deforming a coarsely specified initial curve until it fits along an edge or ridge in the image, using an energy minimisation approach. These could allow taxiline shapefiles to be generated much more rapidly, since a human operator could create a rough initial curve which would then fit itself to a nearby taxiline, rather than having to create a piecewise-linear path with every node placed precisely by hand.

Although we have succeeded in our aim of assessing the effectiveness of many techniques, and produced useful results with several of them, automating the entire AMDB update process remains a distant goal at this stage. To maximise time savings, it may be wise to focus research efforts on detecting the features which are most time consuming for human operators to find and compare. Painted markings on runways, taxiways and aprons are likely candidates, since their small size makes them both harder for humans to locate and easier for image processing techniques to recognise. This is because they are morphologically the simplest objects in the image and a wealth of scientific literature on the subject of curvilinear extraction already exists, in contrast with larger objects such as whole taxiways, which are more complex and less well researched. Painted markings are also likely to change more often than other objects which require significant construction effort to modify.

# Bibliography


[1] P. T. Jackson, C. Nelson, J. Schiefele, and B. Obara, "Runway detection in high resolution remote sensing data", in *International Symposium on Image and Signal Processing and Analysis*, Zagreb, Croatia, 2015, pp. 170–175.

[2] J. Han, L. Guo, and Y. Bao, "A method of automatic finding airport runways in aerial images", in *International Conference on Signal Processing*, vol. 1, 2002, pp. 731–734.

[3] S. Tandra and Z. Rahman, "Robust edge-detection algorithm for runway edge detection", in *Proceedings of SPIE*, vol. 6813, 2008.

[4] V. Kniaz, "A fast recognition algorithm for detection of foreign 3d objects on a runway", *ISPRS-International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 1, pp. 151–156, 2014.

[5] Z. Yang, J. Zhou, and F. Lang, "Detection algorithm of airport runway in remote sensing images", *TELKOMNIKA Indonesian Journal of Electrical Engineering*, vol. 12, no. 4, pp. 2776–2783, 2014.

[6] O. Aytekin, U. Zongur, and U. Halici, "Texture-based airport runway detection", *IEEE Geoscience and Remote Sensing Letters*, vol. 10, no. 3, pp. 471–475, 2013.

[7] A Huertas, W Cole, and R Nevatia, "Detecting runways in complex airport scenes", *Computer Vision, Graphics, and Image Processing*, vol. 51, no. 2, pp. 107–145, 1990.

[8] N. Di, M. Zhu, and Y. Wang, "Real time method for airport runway detection in aerial images", in *Audio, Language and Image Processing*, 2008, pp. 563–567.

[9] P. V. Hough, "Machine analysis of bubble chamber pictures", in *International Conference on High Energy Accelerators and Instrumentation*, vol. 73, 1959, pp. 554–558.

[10] J. Canny, "A computational approach to edge detection", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 8, no. 6, pp. 679–698, 1986.

[11] Federal Aviation Administration, "Aeronautical information manual", in. 2014, ch. 2.3.

[12] P.-N. Tan, M. Steinbach, and V. Kumar, *Introduction to Data Mining, (First Edition)*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2005.

[13] M. Sezgin and B. Sankur, "Survey over image thresholding techniques and quantitative performance evaluation", *Journal of Electronic Imaging*, vol. 13, no. 1, pp. 146–168, 2004.

[14] D. G. Lowe, "Object recognition from local scale-invariant features", in *IEEE International Conference on Computer Vision*, vol. 2, 1999, pp. 1150–1157.

[15] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-up robust features (surf)", *Computer vision and image understanding*, vol. 110, no. 3, pp. 346–359, 2008.

[16] E. Rosten and T. Drummond, "Fusing points and lines for high performance tracking", in *IEEE International Conference on Computer Vision*, vol. 2, 2005, pp. 1508–1515.

[17] C. Harris and M. Stephens, "A combined corner and edge detector.", in *Alvey Vision Conference*, vol. 15, 1988, p. 50.

[18] S. Leutenegger, M. Chli, and R. Y. Siegwart, "Brisk: binary robust invariant scalable keypoints", in *IEEE International Conference on Computer Vision*, 2011, pp. 2548–2555.

[19] X. Dai and S. Khorram, "The effects of image misregistration on the accuracy of remotely sensed change detection", *IEEE Transactions on Geoscience and Remote Sensing*, vol. 36, no. 5, pp. 1566–1577, 1998.

[20] L. G. Brown, "A survey of image registration techniques", *ACM Comput. Surv.*, vol. 24, no. 4, pp. 325–376, 1992.

[21] H.-M. Chen, P. Varshney, and M. Arora, "Performance of mutual information similarity measure for registration of multitemporal remote sensing images", *IEEE Transactions on Geoscience and Remote Sensing*, vol. 41, no. 11, pp. 2445–2454, 2003.

[22] K. P. Murphy, *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.

[23] K. Besbes and R. Bouchiha, "Automatic remote-sensing image registration using surf", *International Journal of Computer Theory and Engineering*, vol. 5, no. 1, pp. 88–92, 2013.

[24] M. Teke and A. Temizel, "Multi-spectral satellite image registration using scale-restricted surf", in *International Conference on Pattern Recognition*, 2010, pp. 2310–2313.

[25] R. J. Radke, S. Andra, O. Al-Kofahi, and B. Roysam, "Image change detection algorithms: a systematic survey", *IEEE Transactions on Image Processing*, vol. 14, pp. 294–307, 2005.

[26] M. Hussain, D. Chen, A. Cheng, H. Wei, and D. Stanley, "Change detection from remotely sensed images: from pixel-based to object-based approaches", *{ISPRS} Journal of Photogrammetry and Remote Sensing*, vol. 80, no. 0, pp. 91–106, 2013.

[27] M.-L. Nordberg and J Evertson, "Vegetation index differencing and linear regression for change detection in a swedish mountain range using landsat tm® and etm+® imagery", *Land Degradation & Development*, vol. 16, no. 2, pp. 139–149, 2005.

[28] F. Yuan, K. E. Sawaya, B. C. Loeffelholz, and M. E. Bauer, "Land cover classification and change analysis of the twin cities (minnesota) metropolitan area by multitemporal landsat remote sensing", *Remote Sensing of Environment*, vol. 98, no. 2-3, pp. 317–328, 2005.

[29] A. Miller, E. Bryant, and R. Birnie, "An analysis of land cover changes in the northern forest of new england using multitemporal landsat mss data", *International Journal of Remote Sensing*, vol. 19, no. 2, pp. 245–265, 1998.

[30] A. Singh, "Review article digital change detection techniques using remotely-sensed data", *International Journal of Remote Sensing*, vol. 10, no. 6, pp. 989–1003, 1989.

[31] O. Miller, A. Pikaz, and A. Averbuch, "Objects based change detection in a pair of gray-level images", *Pattern Recognition*, vol. 38, no. 11, pp. 1976–1992, 2005.

[32] R. Achanta, S. Hemami, F. Estrada, and S. Susstrunk, "Frequency-tuned salient region detection", in *IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 1597–1604.

[33] I. Katramados and T. Breckon, "Real-time visual saliency by division of gaussians", in *IEEE International Conference on Image Processing*, 2011, pp. 1701–1704.

[34] Y. Hsu, H.-H Nagel, and G Rekers, "New likelihood test methods for change detection in image sequences", *Computer Vision, Graphics, and Image Processing*, vol. 26, no. 1, pp. 73–106, 1984.

[35] R. Radke. [Online]. Available: https://www.ecse.rpi.edu/~rjradke/research.htm.

[36] K. Skifstad and R. Jain, "Illumination independent change detection for real world image sequences", *Computer Vision, Graphics, and Image Processing*, vol. 46, no. 3, pp. 387–399, 1989.

[37] E. Durucan and T. Ebrahimi, "Change detection and background extraction by linear algebra", *Proceedings of the IEEE*, vol. 89, no. 10, pp. 1368–1381, 2001.

[38] T. Blaschke, "Object based image analysis for remote sensing", *{ISPRS} Journal of Photogrammetry and Remote Sensing*, vol. 65, no. 1, pp. 2–16, 2010.

[39] G. Chen, G. J. Hay, L. M. T. Carvalho, and M. A. Wulder, "Object-based change detection", *International Journal of Remote Sensing*, vol. 33, no. 14, pp. 4434–4457, 2012.

[40] C. Kirbas and F. Quek, "A review of vessel extraction techniques and algorithms", *ACM Computing Surveys (CSUR)*, vol. 36, no. 2, pp. 81–121, 2004.

[41] A. R. Khan, A. Cornea, L. A. Leigland, S. G. Kohama, S. N. Jespersen, and C. D. Kroenke, "3d structure tensor analysis of light microscopy data for validating diffusion {mri}", *NeuroImage*, vol. 111, pp. 192–203, 2015.

[42] M. Fischler, J. Tenenbaum, and H. Wolf, "Detection of roads and linear structures in low-resolution aerial imagery using a multisource knowledge integration technique", *Computer Graphics and Image Processing*, vol. 15, no. 3, pp. 201–223, 1981.

[43] A. F. Frangi, W. J. Niessen, K. L. Vincken, and M. A. Viergever, "Multiscale vessel enhancement filtering", in *Medical Image Computing and Computer-Assisted Intervention*, 1998, pp. 130–137.

[44] A. Enquobahrie, L. Ibanez, E. Bullitt, and S. Aylward, "Vessel enhancing diffusion filter", *The Insight Journal*, 2007.

[45] T. Jerman, F. Pernuš, B. Likar, and Ž. Špiclin, "Beyond frangi: an improved multiscale vesselness filter", in *SPIE Medical Imaging*, 2015, 94132A–94132A.

[46] L. Antiga, "Generalizing vesselness with respect to dimensionality and shape", *The Insight Journal*, vol. 3, 2007.

[47] A. M. Mendonca and A. Campilho, "Segmentation of retinal blood vessels by combining the detection of centerlines and morphological reconstruction", *IEEE Transactions on Medical Imaging*, vol. 25, no. 9, pp. 1200–1213, 2006.

[48] S. Chaudhuri, S. Chatterjee, N. Katz, M. Nelson, and M. Goldbaum, "Detection of blood vessels in retinal images using two-dimensional matched filters", *IEEE Transactions on Medical Imaging*, vol. 8, no. 3, pp. 263–269, 1989.

[49] E. Ricci and R. Perfetti, "Retinal blood vessel segmentation using line operators and support vector classification", *IEEE Transactions on Medical Imaging*, vol. 26, no. 10, pp. 1357–1365, 2007.

[50] A. Hoover, V. Kouznetsova, and M. Goldbaum, "Locating blood vessels in retinal images by piecewise threshold probing of a matched filter response", *IEEE Transactions on Medical Imaging*, vol. 19, no. 3, pp. 203–210, 2000.

[51] K. Deguchi, T. Izumitani, and H. Hontani, "Detection and enhancement of line structures in an image by anisotropic diffusion", *Pattern Recognition Letters*, vol. 23, no. 12, pp. 1399–1405, 2002.

[52] W. Freeman and E. Adelson, "The design and use of steerable filters", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, no. 9, pp. 891–906, 1991.

[53] E. R. Dougherty, R. A. Lotufo, and T. I. S. for Optical Engineering SPIE, *Hands-on morphological image processing*. SPIE press Bellingham, 2003, vol. 71.

[54] D. D. Lewis, "Naive (bayes) at forty: the independence assumption in information retrieval", in *Machine learning: ECML-98*, Springer, 1998, pp. 4–15.

[55] J. E. Bresenham, "Algorithm for computer control of a digital plotter", *IBM Systems Journal*, vol. 4, no. 1, pp. 25–30, 1965.

[56] A. C. Ruifrok and D. A. Johnston, "Quantification of histochemical staining by color deconvolution.", *International Academy of Cytology [and] American Society of Cytology*, vol. 23, no. 4, pp. 291–299, 2001.

[57] P. Perona and J. Malik, "Scale-space and edge detection using anisotropic diffusion", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, no. 7, pp. 629–639, 1990.

[58] C. Steger, "An unbiased detector of curvilinear structures", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 2, pp. 113–125, 1998.

[59] T. Wagner, *Ridge detection*. [Online]. Available: http://fiji.sc/Ridge_Detection.

[60] L. Vincent, "Morphological grayscale reconstruction in image analysis: applications and efficient algorithms", *IEEE Transactions on Image Processing*, vol. 2, no. 2, pp. 176–201, 1993.

[61] M. Kass, A. Witkin, and D. Terzopoulos, "Snakes: active contour models", *International Journal of Computer Vision*, vol. 1, no. 4, pp. 321–331, 1988.